分类号：**TP391** 单位代码：**10422**

密　级： 学　　号：**200912987**

博士学位论文

Dissertation for Doctoral Degree

论文题目：**基于点缓存全局光照技术的研究**

RESEARCH ON POINT BASED GLOBAL ILLUMINATION

作　者　姓　名　王贝贝

培　养　单　位　计算机科学与技术学院

专　业　名　称　计算机软件与理论

指　导　教　师　孟祥旭 教授

合　作　导　师　Tamy Boubekeur 教授

2014 年 11 月 15 日

RESEARCH ON POINT BASED GLOBAL ILLUMINATION

By

Beibei WANG

Supervisor: Prof. Xiangxu MENG

Co- Supervisor: Prof. Tamy BOUBEKEUR

For the degree of

Doctor of Philosophy

Shandong University, Jinan, Shandong, P.R.China

November 2014

原 创 性 声 明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律责任由本人承担。

论文作者签名：_____ 日 期：_____

关于学位论文使用授权的声明

本人完全了解山东大学有关保留、使用学位论文的规定，同意学校保留或向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权山东大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存论文和汇编本学位论文。

（保密论文在解密后应遵守此规定）

论文作者签名：_____导师签名：_____日 期：_____

目 录

TABLE OF CONTENTS

摘 要

随着动漫产业的发展，人们对真实感渲染的要求越来要高，全局光照做为真实感渲染中的重要组成部分，一直被广泛关注。全局光照既需要计算从光源直接得到的光照，还需要计算经过场景中物体反弹之后的光照，该问题描述为渲染方程的求解。目前常用的计算全局光照的算法包括了蒙特卡洛光线跟踪算法、光子映射、多光源算法以及基于点的全局光照算法等。 本文对基于点的全局光照算法 (Point based global illumination， 简称为 PBGI) 展开研究。

PBGI 算法主要用于模拟环境光遮挡、色溢、面积光以及光泽反射等效果，虽然不能保证结果的正确性，但是渲染效果是无噪声，并且与光线跟踪相比，具有更高效率。 该算法分为两个阶段：第一阶段，对场景进行采样并且计算每个采样点的直接光照，形成点云，将点云组织到空间层次结构 (比如 Bounding Sphere Hierarchy，简称为 BSH) 中，该点云层次结构中的每个节点，表示其子树的平均位置、法向以及光照信息。 第二阶段，利用点云层次结构计算着色点的间接光照，表里点云层次结构，找到对当前着色点可能产生贡献的节点，根据节点距离着色点距离的不同采用不同的策略，距离较远的节点，无需遍历到其中的每一个点， 将这些节点投影到为该着色点构建的微缓冲区 (包含节点深度和颜色信息的半球缓冲区或者立方体缓冲区)，计算这些节点的可见性，最后将微缓冲区中的颜色值，与其对应的双向反射分布函数(Bidirectional Reflection Distribution Function，简称为 BRDF)，以及立体角等卷积后得到该着色点的间接光照值。 PBGI 的高效性以及无噪声是它的主要优势，但是 PBGI 算法也存在一些问题，比如无法计算非漫反射光线传递，即不能模拟焦散和光泽反射表面之间的反射等效果，而这些效果在全局光照中至关重要，因此该问题限制了 PBGI 算法的应用范围。在本文中，我们针对 PBGI 中存在的问题展开研究，一方面提高其效率，另一方面扩展该算法使其支持更多光线传递路径，另外，我们对 PBGI 算法应用到预览渲染中进行了初步探索。

本文提出了基于分解的点缓存全局光照算法，考虑到着色点之间的空间连贯性，通过重用树切以及微缓冲区来提高效率。 在 PGBI 算法中，计算每个着色点

间接光照的过程是独立的，但是我们发现位置和法向相近的着色点具有相似的树切，即这些着色点的遍历过程存在冗余。我们首先提出着色点关于位置和法向的相似性模型，并且基于 k-means 聚类算法将着色点进行聚类，在每个聚类中找到聚类树切，并且将聚类树切中的节点分为远节点和近节点：远节点被聚类中的着色点所共用并且不再进行遍历，而近节点分别被每个着色点进一步遍历。此外，我们使用聚类微缓冲区计算远节点的可见性，着色点微缓冲区计算近节点以及更新节点的可见性，通过将这两种微缓冲区融合得到入射辐射亮度，与 BRDF 卷积后得到间接光照值。该算法对树切进行重用，从而使得 PBGI 的效率提高数倍，并且产生的误差较少，此外，保持了原算法的时间连贯性。

我们提出了基于小波的 PGBI 算法计算非漫反射表面之间的光线传递。在 PBGI 算法中，使用 RGB 存储点的直接光照，使用球谐函数 (Spherical Harmonics，简称为 SH) 系数来存储点云层次结构中的直接光照，由于只存储了漫反射点的直接光照，因此无法模拟焦散等效果。如果采用 SH 来存储非漫反射点的光线传递，低频 BRDF 能够使用较少的系数来表示，而高频 BRDF 则需要大量的 SH 系数，由于 SH 不支持非线性近似，将造成严重的内存问题，而哈尔小波支持非线性近似，具有稀疏性，在内存使用方面具有优势；此外，在表示高频函数时，SH 存在“ringing”走样。基于以上原因，本文中使用哈尔小波来存储直接光照，即小波系数缓存点的出射辐射亮度，在立方体贴图上采样后，在每个面上进行二维小波变换来得到小波系数，并且将小波系数在点云层次结构中进行层次化编码，即将小波系数再次进行小波分析，在点云层次结构的每个节点中计算节点近似系数和节点细节系数，对于高层的节点 (远离根节点的节点层数越大) 只存储节点细节系数，低层的节点只存储节点近似系数，从而减少内存占用。此外，在计算点云层次结构的节点小波系数时，采用后序遍历的策略来减少内存使用。PBGI 在高频光照或者高频 BRDF 情况下会出现走样问题，这是由于对光照或者 BRDF 采样不足造成的，该问题可以通过提高微缓冲区的分辨率来解决，但是这将严重影响到渲染效率，使得 PBGI 失去原有速度上的优势，因此我们提出了重要性驱动缓冲区技术，只在“重要”方向增加微缓冲区的分辨率。根据 BRDF 和入射光照重要性构造重要性函数，细分重要性高的微缓冲区像素，通过实验证明，该重要性驱动微缓冲区既解决了走样问题，又保持了算法的高效性。基于小波的 PBGI 算

法，能够计算从漫反射到非常接近于镜面反射的各个频率的材质，并且与双向路径算法相比有更高的效率。

在以上两个算法的基础上，我们提出基于视点树的多次反射计算方法，用于快速计算光线在场景中的多次反射。 PBGI 算法中，在计算多次反射时，需要计算点云中每个点的间接光照， 即每个点都需要遍历点云层次结构以及向微缓冲区投影树切中的节点， 得到入射辐射亮度之后，再根据 BRDF 计算出射辐射亮度。基于空间连贯性，我们提出利用视点树进行遍历的方案， 即点云树中的某些节点对于其他某些节点中包含点的贡献是相似的， 此时，没有必要对这些点分别遍历来找到该节点。 在多次反射中，点云树有两个作用：发射和接收，为了表达的清晰，分别称为点云树和视点树， 基于视点树遍历的目标是尽可能找到反射节点和接收节点对，使得反射节点对接收节点中所有点的贡献是相似的， 通过计算两个立体角来达到该目标。 最终，重用的树切分布在该视点树中，这是对基于分解的 PBGI 算法的扩展，从聚类内着色点的树切重用扩展到层次化地重用。 此外，在多次反射中的另外一个问题是如何由入射辐射亮度快速得到出射辐射亮度。 假设出射方向和入射方向都在 $n \times n$ 的半球 (投影为正方形) 上采样，即需要计算 n^2 次 BRDF 与入射辐射亮度的卷积， 该问题的时间复杂性 $O(n^4)$。 根据小波稀疏性的特点，我们分别将入射辐射亮度和 BRDF 进行小波变换， 然后在频率空间相乘，从而提高效率。本文提出了 BRDF 四维小波系数和入射光照二维小波系数相乘快速计算出射辐射亮度的模型。 最后，我们利用缓存了多次反射光照的点云层次结构，充分使用图形处理单元并行计算的优势，实现场景的预览渲染，首先将点云层次结构中包含节点近似系数的某一层节点作为采样点，将这些采样点向屏幕空间进行投影，得到采样点对每个像素所包含着色点的辐射亮度以及权重，并且将它们累加到两个缓冲区中， 再根据这两个缓冲区计算出着色点的辐射亮度。场景中可以包含漫反射材质以及各种频率的光泽反射材质，并且支持视点的变化。

本文对 PBGI 算法进行改进和扩展，一方面使其具有更高效率，另一方面，使其支持非漫反射光线传递，从而可以模拟焦散等效果，扩大其应用范围。

关键词： 全局光照，基于点缓存的全局光照，真实感渲染，哈尔小波，非漫反射光线传递

ABSTRACT

As the development of animation industry, there are more and more requirements for the realistic rendering. Global illumination (GI) is an significant part of realistic rendering, and it has been focused by researchers for many years. Not only the lighting directly from the light sources but also the lighting reflected by other objects in the scene is required to be computed, so it's complicated to solve the GI problem that can be described as the rendering equation. Several algorithms can be used to solve this problem, such as Monte Carlo based ray tracing, photon mapping, many lights based approaches, point based global illumination and so on. This thesis is about Point Based Global Illumination (PBGI). PBGI is a popular rendering algorithm in movie and motion picture productions. This algorithm provides a diffuse global illumination solution by caching radiance in a mesh-less hierarchical data structure during a pre-process, while solving for the visibility over this cache, at rendering time, for each receiver, using a microbuffer, which is a localized depth and color buffer inspired from real time rendering environments. As a result, noise free ambient occlusion, indirect soft shadows and color bleeding effects are computed efficiently for high resolution image output and in a temporally coherent fashion. PBGI has attracted increasing attention nowadays because of its efficiency and noise free quality. However, there are still some problems, such as it can not simulation non-diffuse light transport, that makes it have limited applications. My thesis aims to solve these issues in PBGI and extend it to support more light transport path.

Based on the spatial coherency, we propose a factorized solution of PBGI to make it more efficient by reusing the tree cut and the microbuffers. In PBGI, each receiver traverse the point cloud tree independently, but we observe that the similar receivers have the similar tree cut, that means there is redundancy during the traversal process. A similar model of the receivers is proposed at first, and then it is used to cluster the receivers. The point cloud tree is traversed for a cluster instead of for each receiver, and a cluster tree cut is obtained. The far nodes in the cluster tree cut are shared directly by all the receivers in

this cluster without further traversing, while the near nodes are traversed for each receiver independently. A cluster microbuffer is proposed to solve for the visibility of the far nodes, while the receiver specific microbuffer is used to solve for the visibility of the near nodes and the refined nodes. The final microbuffer by combining these two microbuffers is convolved with the bidirectional reflection distribution funcion (BRDF) of the receiver to get the final indirect illumination. Our algorithm offers a significant rendering speed-up for a negligible and controllable approximation error and it inherits the temporal coherence of PBGI.

We also propose a wavelet based solution to PBGI to compute the non-diffuse light transport. As only the diffuse lighting of the point is baked in PBGI, it can not simulate the non-diffuse light transport, such as caustics. PBGI tree nodes uses spherical harmonics (SH) to represent outgoing radiance. Unfortunately, even using a larger number of coefficients, SH are not able to capture high frequencies efficiently, which translates in our case to non-diffuse reflections or refractions. Consequently, caustics stemming from metals, plastics, glass and other reflective or refractive materials are not handled with classical PBGI frameworks. Even when ignoring the performance issue induced by a larger number of SH coefficients, ringing artifacts quickly appear. Compared with SH, haar wavelets support non-linear approximation, so the the representation is compact. So we propose to represent the outgoing radiance of the non-diffuse point with wavelet coefficients by sampling according to a cube map firstly and wavelet transforming each face of this cube map. The coefficients are further encoded hierarchically in the point cloud tree to decrease the memory usage, that means the coefficients themselves are wavelet transformed, generating two kinds of coefficients: node approximation coefficients and node detail coefficients. The node approximation coefficients are stored for the low level nodes (close to the root), and the node detail coefficients are stored for the high level nodes. To avoid storing the entire list of nodes vectors at any intermediate state, we compute this compressed representation during a post-order depth-first traversal of the PBGI tree. Further more, according to the artifacts problem that appears when there is high frequency BRDF or lighting in the scene, we propose to use the importance driven microbuffer. The importance function

that includes the incoming lighting importance and the BRDF importance is used to drive the microbuffer, that means when one pixel has high frequency information (lighting or BRDF), it will be subdivided. Finally, our rendering algorithm allows to handle non-diffuse light transport, reproducing caustics with a similar quality to bidirectional path tracing for only a fraction of the computation time, with an intuitive control on the approximation error.

Based on the previous two algorithms, we propose a view-tree based approach to compute the multiple bounces reflection. In PBGI, the indirect illumination of each point in the point cloud needs to be evaluated by traversing the point cloud tree and splatting the nodes in the tree-cut, so each point is treated a receiver. We propose to organize all the receivers into a view tree, and the point cloud tree is traversed for the view tree instead of for each receiver. The view tree approach is based on the observation some nodes in the point cloud tree contribute similarly to all the points in other nodes, that means we don't need to traverse for these points respectively but only for the node that contains these points. This is an extension to the factorized PGBI from one level (cluster) to a hierarchical structure (tree). Another problem for multiple bounces computation is how to evaluate the outgoing radiance from the incoming radiance efficiently. As the outgoing radiance needs to be computed for each sampling direction, the time complexity is $O(n^4)$, where $n \times n$ represents the resolution of the hemisphere or square according to which the incoming direction and the outgoing direction are sampled. The wavelet representation is sparse that improves the performance, so we decide to wavelet transform the incoming radiance and the BRDF and multiply them in the frequency domain. We propose a novel outgoing radiance computation model by doing product between 4D BRDF wavelet coefficients and 2D incoming radiance wavelet coefficients. Finally, the point cloud tree with multiple bounces reflection stored is used to offer a preview rendering of the scene by utilizing the GPU computing efficiently. We can support scenes that include diffuse materials and all frequency glossy materials with a changing camera.

Our thesis improves and extends the PGBI algorithm so that it can be used in more applications.

Keywords: global illumination, point based global illumination, realistic rendering, haar wavelet, non-diffuse light transport

第一章　引言

图 1.1　图中作品依次为：左上图为梦工厂的《功夫熊猫2》，右上图为迪斯尼的《怪兽大学》，左下图为迪斯尼的《飞屋环游记》，右下图为梦工厂的《马达加斯加3》。

随着电影、动漫产业的发展，人们对真实感渲染的需求也越高，全局光照作为真实感渲染中的重要组成部分，得到越来越多的关注。图 1.1 中列举了几部采用全局光照技术的动漫作品。

简单来讲，全局光照是指既考虑场景中直接来自光源的光照又考虑经过场景中其他物体反射后的光照。全局光照问题具有递归性、全局性特点，并且需要进行积分，可以抽象为绘制方程的求解问题，计算的复杂性决定了解决该问题很耗时。计算全局光照的主要算法包括蒙特卡洛光线跟踪、光子映射、多光源以及基于点缓存的全局光照算法等，我们将在本章进行介绍这些算法及各自的优缺点。本文的研究对象是基于点的全局光照算法 (或者称为基于点缓存的全局光照算法)，该算法由于高效性和无噪声的优势在提出之后得到了大量应用，主要用于模拟色溢、环境光遮挡以及环境映射等效果，但是该算法并不能模拟某些特效，比如焦

散等， 因此本文对该算法进行研究，解决其中存在的问题，并且保持它在效率上的优势，从而使其具有更广泛的应用。

在本节中，我们首先介绍全局光照相关的基本概念和基本理论，然后介绍主要的全局光照算法， 最后介绍本文的研究工作概要。

1.1 相关理论

1.1.1 假设

本文中光线传递采用几何光学模型，即光线只有在表面才会被发射、散射和吸收，在表面之间按照直线传播。 此外，我们忽略了光的波动性，比如衍射等。在一般情况下，这些忽略的效果并不重要，并且几何光学模型能够满足常见模型，因此，很多渲染算法做出相同的假设。

1.1.2 双向反射分布函数和双向散射分布函数

双向反射分布函数 (bidirectional reflectance distribution function， 简称为 BRDF) [1] 是定义在表面上某点的关于入射光线反方向和出射光线方向的六维函数， 定义如下：

$$\rho(x,\omega_i,\omega_o)=\frac{dL_r(x,\omega_o)}{dE_i(x,\omega_i)}=\frac{dL_r(x,\omega_o)}{L_i(x,\omega_i)(\omega_i\cdot n)d\omega_i}.$$

其中 ρ 表示 BRDF 值，x 表示反射点的位置，ω_i 表示入射光线的方向， ω_o 表示出射光线的方向，L_r 表示出射光的亮度， E 表示光的照度，L_i 表示入射光的亮度，n 表示反射点的法向。 BRDF 表示沿着 ω_o 方向的反射辐射亮度在沿着 ω_i 方向上入射辐射照度的比例， 其单位是 sr^{-1}，其中 sr 表示单位立体角。

BRDF 描述了材质表面的各种性质，比如颜色、光泽度、反射程度等，是一种局部光照模型。 若已知入射光线的辐射亮度，则可以计算出任何出射方向的辐射亮度。 BRDF 的一个重要性质是能量守恒，即物体表面反射的光不可能大于接收到的光， 即 BRDF 应该满足以下公式：

$$\int_\Omega (x,\omega_i,\omega_o)(\omega_i,n)d\omega_i\le 1,\ \ \forall\omega_o.$$

BRDF 描述材质的反射特征，同样地，双向透射分布函数 (bidirectional transmittance distribution function，简称为 BTDF) 描述光线的折射特征， 这两种函数统

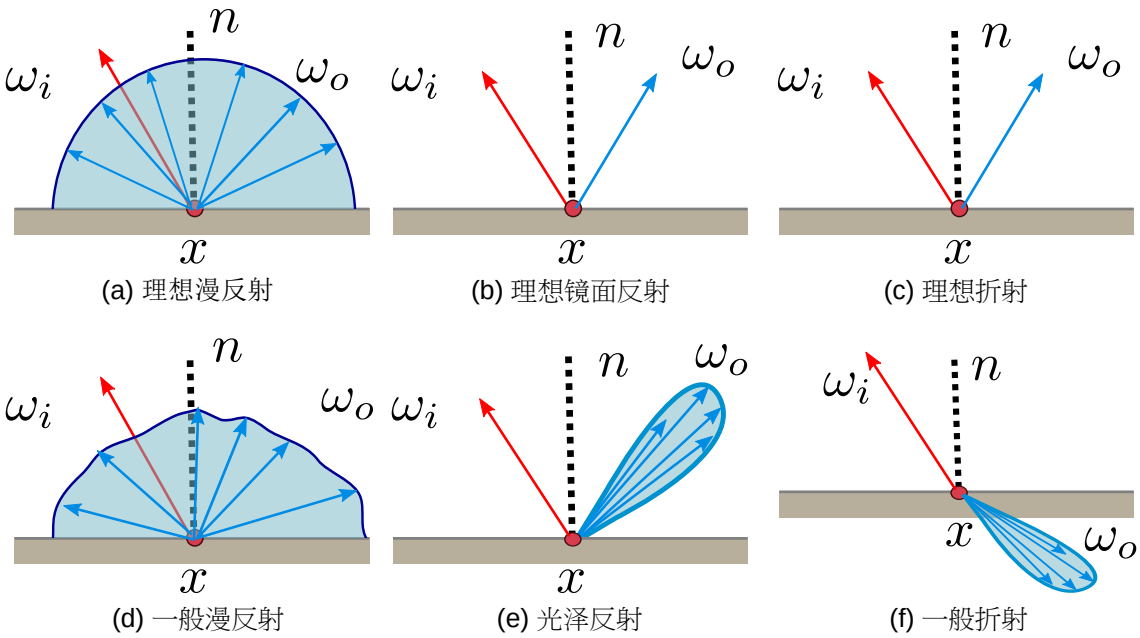


图 1.2 常见的 BSDF 模型

称为双向散射分布函数(bidirectional scattering distribution function，简称为 BSDF)。

图 1.2 中展示了几种常见 BSDF 模型。本文采用的 BSDF 是 Mitsuba Render [2] 中的模型，其中光泽反射和折射，我们采用的是 Rough Conductor 和 Rough Dielectric [3]。

1.1.3 绘制方程

将场景中最终投影到图像中的点称为着色点，或者理解为从像素发射初级光线与场景相交产生的点。全局光照是指除了计算着色点从光源接收到的光外，还要计算被场景中物体反射后到达着色点的光，以及着色点自身所发出的光，如图 1.3 所示。通常使用渲染方程 [4] (rendering equation) 来表示全局光照问题。

$$L(x,\omega_o) = L_e(x,\omega_o) + \int_{\Omega_{2\pi}} L_i(x,\omega_i)\rho(x,\omega_i,\omega_o)(\omega_i \cdot n)d\omega_i, \tag{1.1}$$

其中 L 表示辐射亮度，关于位置 x、出射方向 ω_o 的函数，L_e 表示着色点在出射方向上的自发光，L_i 表示入射辐射亮度，ω_i 表示入射方向，ρ 表示双向反射分布函

(a) 直接光照　(b) 间接光照　(c) 直接光照 + 间接光照

图 1.3 Sponza 场景，在 (a) 中只计算直接光照，(b) 中只计算间接光照，(c) 中计算直接光照和间接光照。

数 BRDF (此处也可以为 BSDF)， n 表示反射点的法向。 该方程表示着色点在某个出射方向上的出射辐射亮度等于该着色点自身在该出射方向上的辐射亮度， 以及半球采样的入射光经过该着色点反射后在该出射方向产生的辐射亮度的和。 计算全局光照的过程实际上是求解绘制方程的过程，方程的递归性和全局性以及存在积分的特点决定了计算全局光照问题的复杂性，我们将在以下的小节中详细介绍求解全局光照的算法。

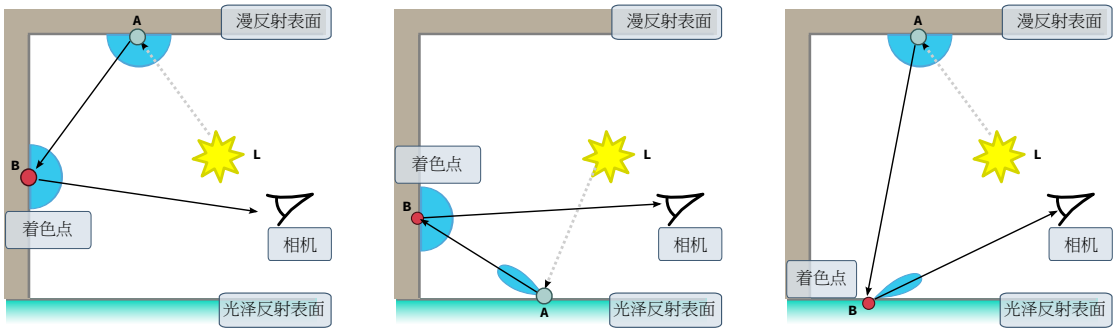
1.1.4 光线传播符号

Heckbert 等人在 [5] 中提出了一系列符号来表示光线在不同表面间的反射或者折射： **L** 表示光源，**E** 表示相机， **S** 表示一次镜面反射 (或者折射)，**D** 表示一次漫反射。 比如 **LSDE** 表示光线从光源发出后，经过镜面反射 (或者折射) 后，与漫反射表面相交， 经过漫反射后到达相机这一路径。

另外，为了描述路径的组合，引入以下的表达式： (k)+ 表示大于等于一次 k 事件，(k)* 表示大于等于零次 k 事件， (k)? 表示零次或者一次 k 事件，(k|k') 表示一次 k 事件或者一次 k' 事件。

1.2 常见的特效

在本节中，我们介绍全局光照中几种常见的特效 (如图 1.4)，比如色溢，光泽反射以及焦散。 在这三种特效中，光线在场景中的传播经过了不同类型的路径。在图 1.4 中，从光源到相机，我们只使用了两条光线，即只有一次反射，但是其路径可以被扩展到多次反射， 以上三种特效可以形式化如下： **LD(D)+E** 表示色



(a) 色溢　(b) 焦散　(c) 光泽反射

图 1.4 光源向场景中发射光线，当光线碰到表面时，其中的部分能量被吸收，其他的被反射。 被反射部分比例以及反射后的颜色和分布 (蓝色表示) 取决于表面的材质 (BRDF)。 漫反射表面的反射光线在半球方向内均匀分布，随着表面的反射度的增加，反射光线的角度越小，直到镜面反射时变成一条线。 当反射光线进入到相机时，其辐射亮度即为像素的颜色。 图中表示了三种不同类型的特效，在 (a) 中，光线经过漫反射表面反射之后到达漫反射表面，然后最终到达相机，即为色溢； 在 (b) 中，光线经过光泽反射表面反射之后再经漫反射表面最后到达相机，即为焦散， 另外经过折射后发生漫反射再到达相机的特效也称为焦散； 在 (c) 中，光线经过漫反射表面反射之后再经过光泽反射表面，最终进入相机，即为光泽反射。

溢，**L(D)+SE** 表示光泽反射，**L(D|S)*S(D|S)E** 表示焦散。 图 1.5 和图 1.6 中举例说明以上三种特效。

1.3 全局光照算法

常见的全局光照算法分为以下几类：蒙特卡洛光线跟踪、辐照度、光子映射、多光源及基于点缓存的全局光照算法。 在本节中，我们对前几种算法进行介绍和分析， 在下一章中，我们将详细介绍基于点缓存的全局光照算法。

1.3.1 蒙特卡洛光线跟踪

传统的光线追踪技术 [7] 从相机向场景中发射光线， 当与完美镜面发射或者完美折射的表面相交时，计算其反射或者折射光线，直到跟踪到光源。 该方法无法计算漫反射表面间的反射，即只能部分求解绘制方程。 路径跟踪算法在 [4] 中提出，通过相机向场景发射光线， 当与场景中的物体相交时，随机的发射反射光

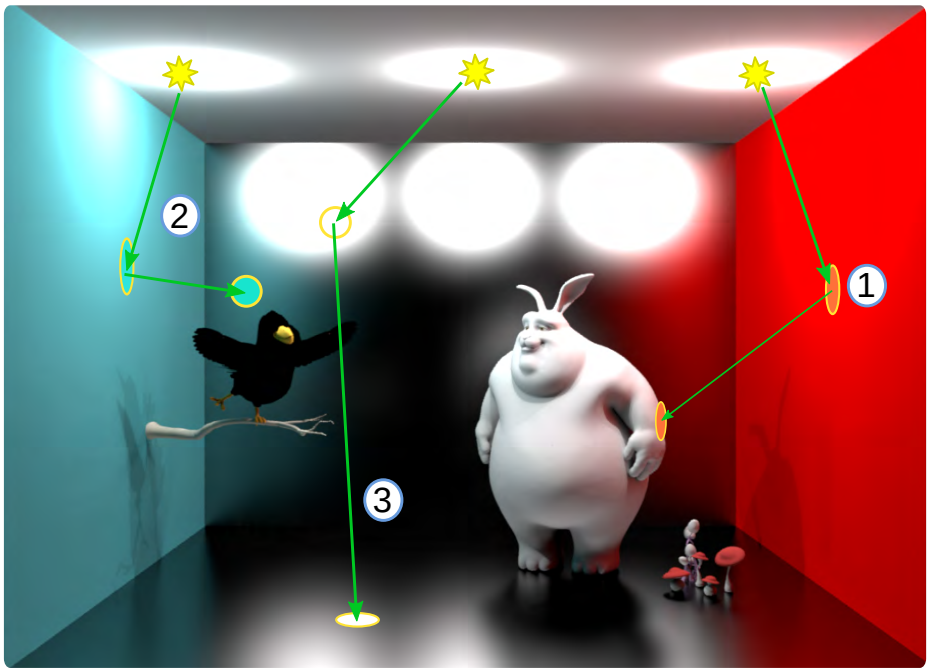


图 1.5 路径1 表示色溢，路径2 表示光泽反射，路径3 表示焦散。

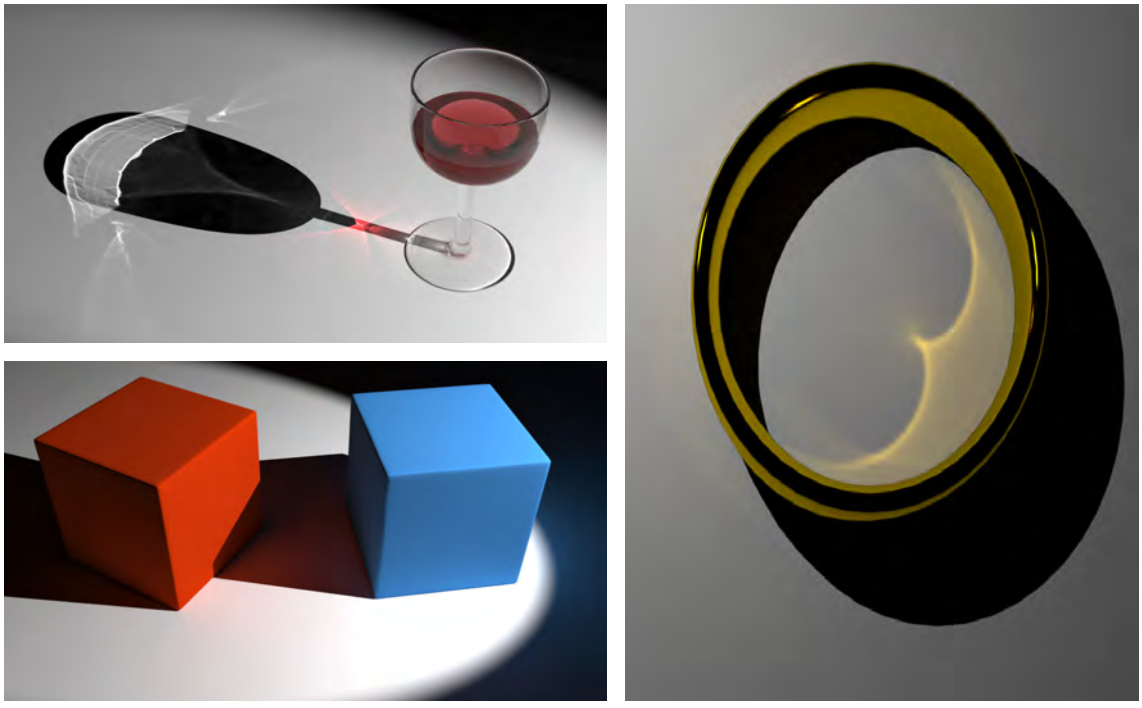
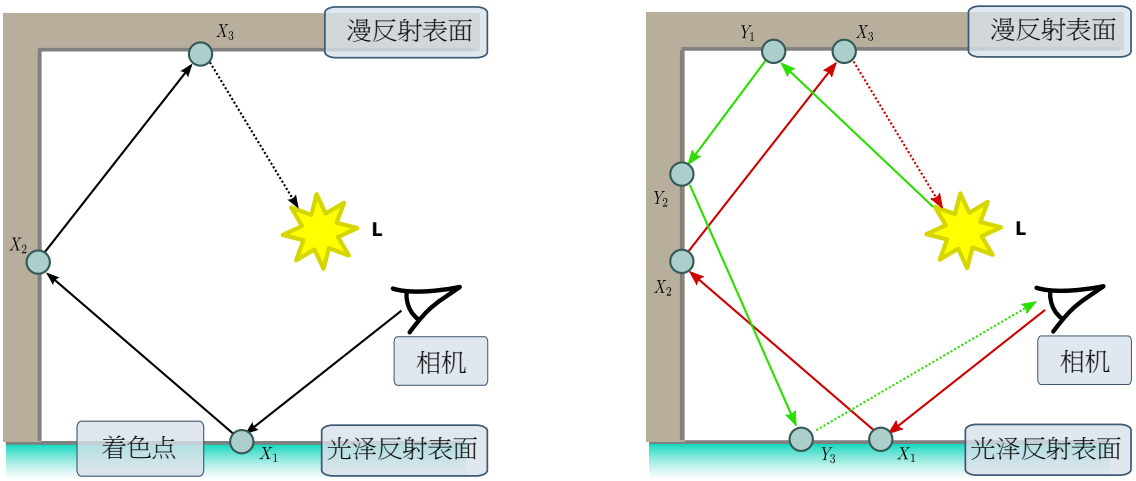


图 1.6 左上图表示折射产生的焦散， 左下图表示色溢，右图表示了反射产生的焦散。(左上和左下图片自于 [6])



(a) 路径跟踪　　(b) 双向路径跟踪

图 1.7 路径跟踪和双向路径跟踪。在路径跟踪中，光线从相机发出，与场景中的物体相交后随机反射，直到与光源相交，而 在双向路径跟踪中，光线分别从相机和光源发出。

线，直到与光源相交，如图 1.7 (a) 所示。 与光线跟踪相比，路径跟踪可以计算任意的材质。该方法的缺陷在于，当场景中有小光源时，光源被光线碰到的可能性比较小，收敛速度较慢， 另外在计算焦散 (**L(S)+DE**) 时，该算法也存在相同的问题。 如果从光源进行采样，可以解决以上问题，于是 [8] 和 [9] 中提出双向路径跟踪 (如图 1.7 (b) 所示)。 双向路径跟踪，从光源和相机开始发射光线，当与场景中物体相交时，随机产生反射光线， 直到与另外一边 (相机和光源) 相交，最后把两个路径组合 (一个路径中的交点向另外一个路径中交点发射阴影光线，并加权平均来计算最终的贡献)。 Veach 等人在 [10] 中提出了 Metropolis light transport (MLT) 算法， 在对最终图像贡献大的路径空间区域进行采样。 该算法与之前的方法相比，最大的区别在于基于一定的分布来产生新的路径，而不是完全随机的， 因此该方法对于某些场景 (光源通过一个小洞来照亮旁边的房间时) 有很大优势，当它找到一条路径时，很容易产生其他的路径。 该方法的缺点是，在辐射亮度很大但是很平滑的区域也进行了大量的采样。 该问题在 [11] 中得以解决，该技术根据图像的梯度来指导采样。 Vorba 等人在 [12] 中提出了基于在线学习的方法来指导重要性采样，在预处理 (训练) 阶段渐进式发射粒子 (光子) 从而得到场景的重要性分

布和辐射亮度分布， 在渲染阶段，利用缓存的两种分布来指导采样，可以用于路径跟踪和双向路径跟踪，从而提高在计算复杂光照时的效率。

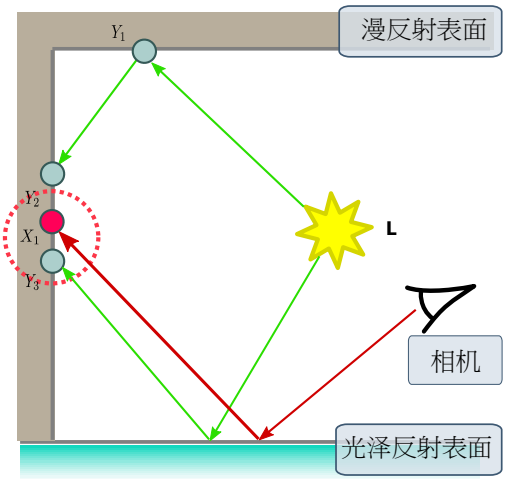
这一类蒙特卡洛方法，具有优点有：第一，内存消耗很小； 第二，可以支持任意的 BSDF； 第三，无偏的；第四，很容易计算镜面反射。 缺点是：对于某些场景需要很长的收敛时间，因此通常可能会有噪声。

1.3.2 辐照度算法

辐射度方法 [13] 先把场景分成面片， 计算面片之间的相互作用 (形式因子)，然后通过迭代法来找到一个光能传递的平衡状态， 该方法只支持漫反射表面， 后来在 [14]， [15] 等中进行了改进， 计算非漫反射表面之间的间接光照，但是仍然无法计算镜面反射。 另外，由于需要计算每两个面片之间的相互作用，因此效率较低， 而 [16] 使用层次结构来组织面片从而提高效率。 辐射度算法的优点是支持计算漫反射表面之间的间接光照。 缺点是，第一，由于需要将场景细分成面片，即对场景的近似表示， 因此不适用于计算尖锐 (sharp) 的特征，比如硬阴影等；第二，需要大量的内存存储面片； 第三，计算复杂模型和非漫反射表面时，很耗时。

1.3.3 光子映射

光子映射在 [17] 中由 Jensen 提出，也是一种蒙特卡罗算法，主要过程如图 1.8 所示。 该方法可以模拟多种全局光照效果，比如色溢现象和焦散。 在较大面积光源或多个镜面反射路径的情况下，光源需要发射大量的光子来避免噪声， 后来在 [18] 中得到改进， 这是一种渐进式的算法，在第一步中进行光线跟踪得到着色点，在之后的多步中进行光子跟踪， 产生新的光子图，并且使用该光子图更新着色点的辐照度值，每次光子跟踪之后，结果会更加精确。 在该方法中，整个光子图不需要保存在内存中，所以可以利用有限的内存达到高精度， 缺点是光子图不可以复用。 Hachisuka 等人在 [19] 提出利用光子路径可见性函数进行重要性采样的技术， 从而使得采样集中在视点可见的范围内。 该技术适用于户外、光线来自于很小的缝隙以及照明区域局部放大等情况下。 该方法的缺点在于只考虑到了可见性的重要性， 没有考虑到 BRDF 的重要性，因此不适用包含高频 BRDF 的场景。



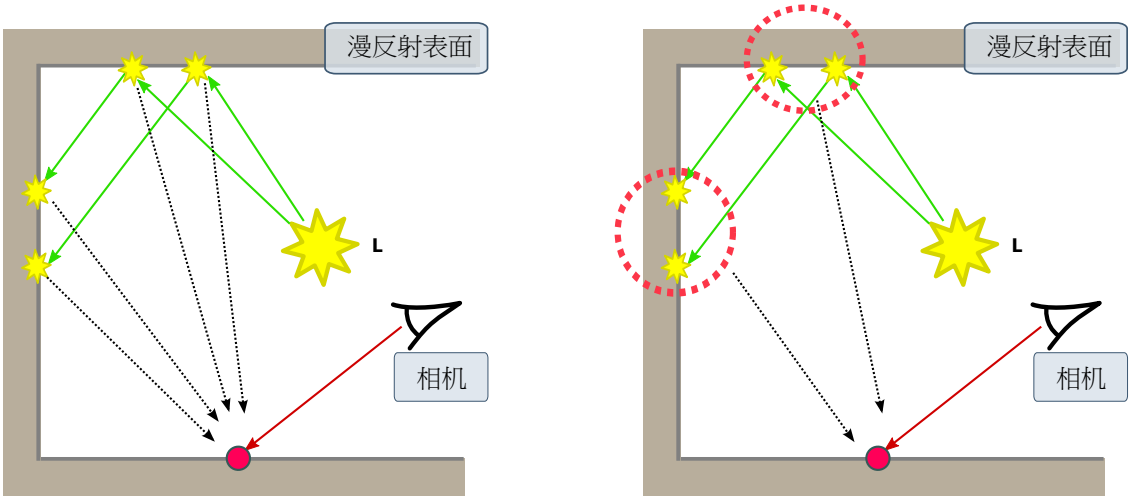
光子映射

图 1.8 光子映射算法。该算法分为两部分： 第一部分，由光源发射光子，与场景相交和反弹，存储与非镜面反射表面的交点形成光子图，如绿色圆圈所示 ；第二部分，从相机发射光线，计算出第一个非镜面的交点 (该过程表示为红色)， 通过对光子图查找 (比如最近邻算法) 计算出该交点的辐射亮度。

光子映射算法的优点在于：能够计算多种全局光照效果，并且具有一致性。缺点在于，同其他蒙特卡罗算法一样，存在噪声问题。

1.3.4 多光源方法

多光源方法最早由 Keller 等人在 [20] 中提出的，称为立即辐照度 (Instant Radiosity，简称为 IR)， 该算法是一种随机算法，并不是与辐照度算法一样的确定性算法。 该算法的过程如图 1.9 所示。 该方法与光子映射不同的地方是，在计算着色点的间接光照时， 前者通过向光源发射光线来计算而后者通过密度估算 (density estimation)。 如果产生的虚拟点光源(Virtual Point Light，简称为 VPL)的个数不多时，也可以为每个 VPL 产生小的阴影图来计算。 IR 的主要问题在于，其质量和速度取决于 VPL 的个数，当场景复杂时，算法可行性降低。 Walter 等人在 [21] 中提出了光源割算法 (light-cuts) 可以应用到 IR 中，将 VPL 组织到层次结构(比如 BSP 树) 中， 在渲染时，对该树进行遍历，相比于每个 VPL 对着色点来计算间接光照，具有更高效率。 由于只有光线与漫反射表面产生的交点才会表示为 VPL， 因此以上算法并不支持非漫反射光线传递。 此外，当点 VPL 距离



(a) 立即辐照度　　(b) 光源割

图 1.9 (a)立即辐照度：第一部分，由光源发射光线，与场景相交形成 VPL(黄色小八角形)；第二部分，对于每个着色点(红色圆圈)，通过光线跟踪 (向每个 VPL 发射阴影光线) 计算间接光照。 (b)光源割，使用了空间层次结构组织 VPL。

某些着色点非常接近 (接近零但是不是零) 时，会产生很强的间接光照， 因此使用 clamp 来防止该类走样，但是会造成光照损失， 该问题在 [22] 中通过使用虚拟球形光源 (VSL) 代替 VPL 得以解决。 Walter 等人在 [23] 中提出双向光源割方法 (Bidirectional Lightcuts) 来模拟焦散， 在该方法中，不仅在第一部分 (产生VPL) 进行递归计算，第二部分 (从相机发射光线) 也进行递归。

该类方法的优点在于将光照分布从场景的几何表示中独立出来，从而更加灵活，这点与光子映射相似。 此外， VPL 的个数可以用于控制在质量和速度之间进行权衡，使其具有广泛的应用，从游戏级的实时渲染到电影级的离线渲染。 与光子映射方法相比，该方法在复杂场景中能更好的计算细节光照，并且噪声更少。该类方法的缺点在于，该方法是有偏的，另外，在计算间接光照的可见性时需要较大代价。

1.4 本文的研究工作以及组织结构

本文针对基于点缓存的全局光照算法中存在的问题展开研究， 从而得到更高效率，更广泛的应用。 本文的研究工作包括了如下：

1. 基于分解的点缓存技术。 针对基于点缓存的全局光照算法中存在的点云树遍历冗余问题，提出利用空间连贯性，提高点云树遍历效率的算法。
2. 基于小波的点缓存技术。 针对基于点缓存的全局光照算法不支持非漫反射光线传递的问题，提出基于小波的策略， 并且提出小波系数层次化编码技术来减少内存使用， 此外，我们提出重要性驱动微缓冲区构建算法，用于高效解决场景中高频光照或者高频 BRDF 带来的走样问题。
3. 高效的多次反射技术及其应用。在前两个研究内容的基础上， 我们提出了基于视点树的多次反射算法，充分利用着色点之间的空间连贯性，提高点云树中间接光照的计算效率。 在更新点云树的出射辐射亮度系数时，我们提出直接使用四维系数与二维系数相乘的方法使得性能得到进一步提高。 最后，将存储了间接光照的点云树用于场景的预览渲染，支持视点可变，并且支持各种频率的光泽反射材质。

本文分为六章，本章介绍基本理论以及主要的全局光照方法，后续的章节安排如下： 第二章介绍基于点缓存的全局光照算法， 第三章介绍基于分解的点缓存全局光照算法， 第四章介绍基于小波的点缓存全局光照算法， 第五章介绍高效的多次反射技术及其应用， 第六章总结本文，并且提出以后的研究方向。

第 二 章　基于点缓存的全局光照

在本章中，我们对点缓存全局光照 (PBGI) 进行介绍和分析。 PBGI 自从在 Christensen [24] 中提出之后，得到了广泛的关注， 被应用于电影和动漫制作中，比如《玩具总动员3》、《飞屋环游记》、《功夫熊猫2》和《怪兽大学》等 (如图 1.1 所示)。 该算法同光线跟踪等算法相比，在模拟色溢、环境光阻挡、软阴影等特效方面具有更高效率 (如图 2.2 所示)。 PBGI 虽然不能得到正确或者无偏的全局光照，但是具有"无噪声"性，而这点对于电影制作至关重要。

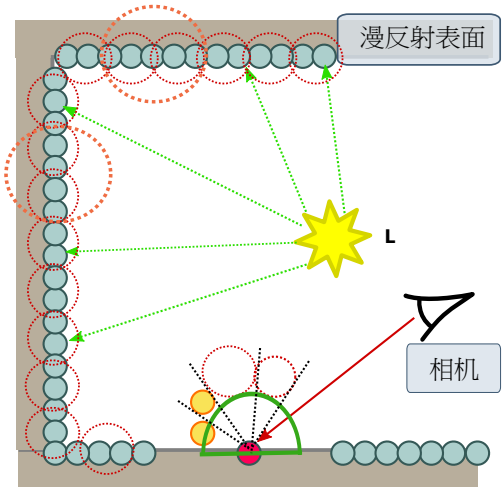
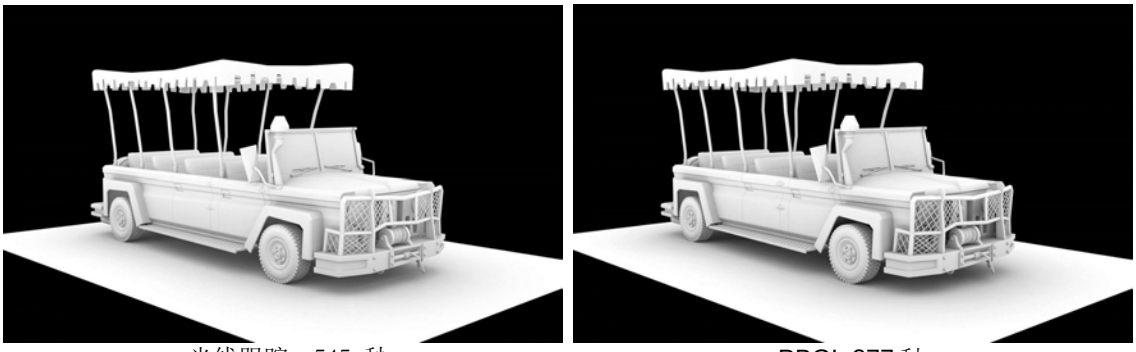


图 2.1 PBGI：第一阶段，对场景采样并且计算采样点 (蓝色圆圈) 的直接光照 (绿色虚线) 形成点云； 将点云组织到空间层次结构 (比如 BSH) (红色虚线圆圈作为节点形成的层次结构) 中。 第二阶段，对于每个着色点 (红色圆圈)，构造微缓冲区 (绿色半圆)，遍历空间层次结构， 将找到的树切中的节点向微缓冲区进行投影 (计算可见性)。 最后将微缓冲区中的颜色值，与其对应的 BRDF 以及立体角等进行卷积后得到该着色点的间接光照值。

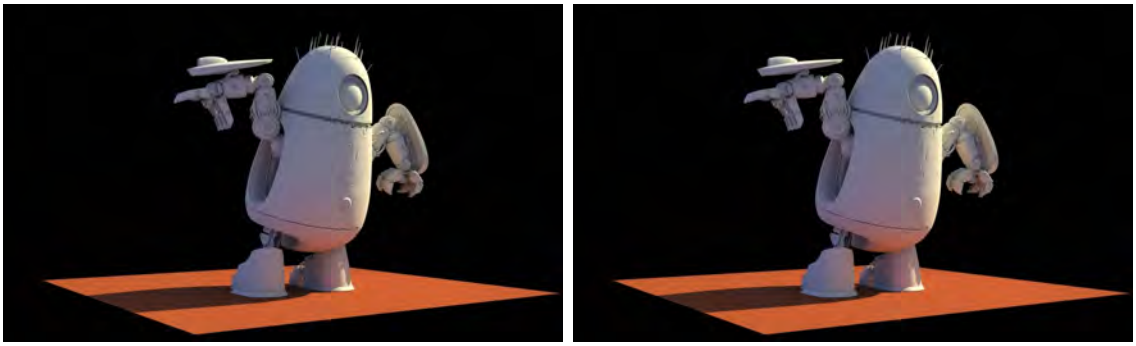
PBGI 分为两个阶段：预处理阶段和渲染阶段，如图 2.1 所示，下面对这两个阶段进行详细介绍。

2.1 预处理阶段

预处理阶段的目的是将直接光照分布在场景的采样点中形成点云，并且将点云组织到点云层次结构中。



光线跟踪，545 秒　　PBGI, 277 秒



光线跟踪，2074 秒　　PBGI, 436 秒

图 2.2　光线跟踪与 PGBI 效率对比。图片与数据来自于 [25]，分别是马达加斯加 (Madagascar: Escape 2 Africa) 与《Monsters vs. Aliens》中的场景。

2.1.1 采样

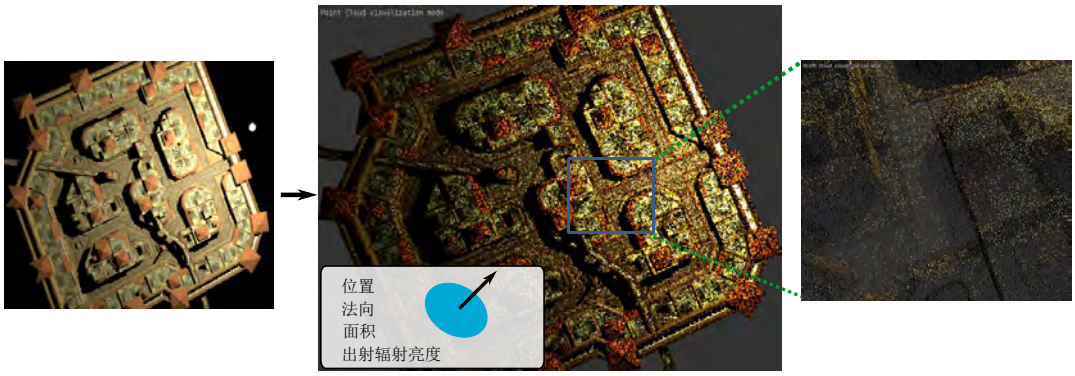


图 2.3 场景 Italian 的点云。

在预处理阶段，首先对场景进行采样。有多种方法可以用于场景的点采样，比如基于累积分布函数 (Cumulative Distribution Function 或者 Inverse Transform

Sampling, 简称为 CDF) 的采样 [26]、 泊松盘采样(Poisson Disk Sampling) [27] [28] [29] 以及基于 REYES [30] 的采样。 基于 CDF 的采样，首先对于场景中三角形的面积计算累计分布函数，然后从该函数中进行均匀采样，最后在每个三角形内部进行均匀采样。 当需要大量采样时，该方法保证了采样点的分布与三角形的面积成正比例； 能够产生均匀的采样，但是不能保证每两个点之间的最小距离。 泊松盘采样，要求每个点之间的最小距离不能小于一个阈值，满足蓝噪声 (blue noise) 性质，产生的效果更好，但是效率低于基于 CDF 的方法。 REYES 算法将场景中的表面细分成微多边形，为每个微多边形着色后，将其作为一个点。

每个点主要包含四个要素：位置，法向，半径以及直接光照（RGB 空间表示）(如图 2.3 所示)。 产生的点云视点无关，因为视点不可见的地方也可能对其他可见的地方产生间接光照。

2.1.2 点云层次结构

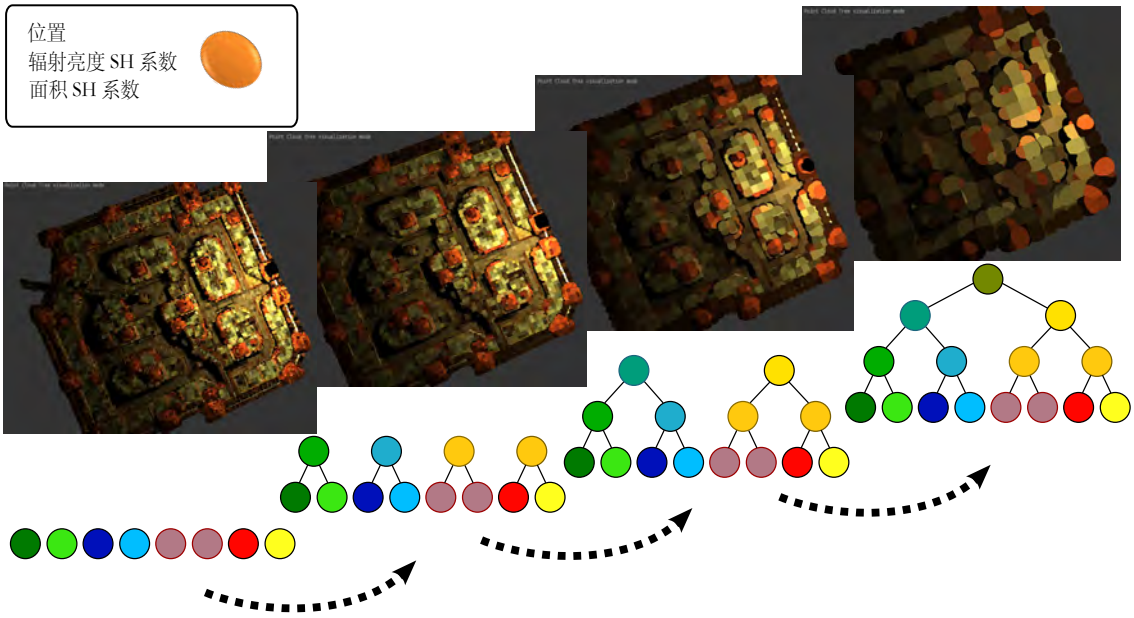


图 2.4 点云层次结构。

场景采样之后，点云中存储了直接光照，点云对着色点产生的影响即为间接光照。 为了避免点云中的每个点对每个着色点直接产生影响，我们将点云组织到一个层次结构中， 从而较远的一簇点一起对着色点产生作用，即层次细节技术

(Level of Detail 简称为 LoD)。 多种结构可以用于组织点云，比如八叉树 [24]，基于外存的八叉树 (out-of-core octree) [31] 以及包围球层次结构 (bounding sphere hierarchy，简称为 BSH) [32] [33] [34]。 图 2.4 中以 BSH 为例展示了点云树的构建过程。 通常该树是自底向上构建的，每个节点中存储了子树的各种属性的平均值，比如位置、直接光照以及半径等。 PBGI 采用球谐函数 (Spherical Harmonics，简称为 SH) 表示节点中的直接光照和面积。 SH 需要少量的系数即可表示低频函数，每个节点，使用度为 3 (即 $L = 3$) 的基函数, 从而产生 9 个系数 c_l^m。 节点的投影面积需要 9 个系数表示；直接光照则需要 27 个系数来表示。 每个点投影面积的计算公式为：

$$A_v(\omega) = \max\left(0, \pi r^2 \langle \mathbf{n}, \omega \rangle\right).$$

投影面积的 SH 系数计算公式为：

$$a_l^m = \int_\Omega A_v(\omega)\, Y_l^m(\omega)\, \mathrm{d}\omega.$$

直接光照的 SH 系数计算公式为：

$$p_l^m = \int_\Omega \mathbf{B} A_v(\omega)\, Y_l^m(\omega)\, \mathrm{d}\omega,$$

其中 **B** 表示漫反射直接光照(通常使用 RGB 表示)。

2.2 渲染阶段

在预处理阶段得到了包含直接光照的点云层次结构后，将其用于计算着色点的全局光照。 在每个着色点处，计算间接光照的主要步骤为：首先对点云层次结构进行遍历， 在遍历过程中使用节点对着色点的立体角来判断是否需要继续遍历， 最后找到可能对当前着色点产生作用的节点，形成节点候选集，称为树切 (tree cut)。 找到树切之后，为每个着色点构造微缓冲区 (包含了深度和颜色属性的缓冲区) 计算节点的可见性， 之所以称为微缓冲区是因为它的分辨率比较小，比如 12×12 的分辨率对于某些场景已经足够。 Christensen 在 [24] 中，使用了平行于全局坐标系坐标轴的立方体缓冲区 (6 个 2D 图片) 作为微缓冲区。 Ritschel 等在 [32] 中使用了半球 (朝向与点法向相同，即局部坐标系) 来作为微缓冲区， 像素与方向之间的映射可以使用标准半球参数化，比如 Nusselt 映射：

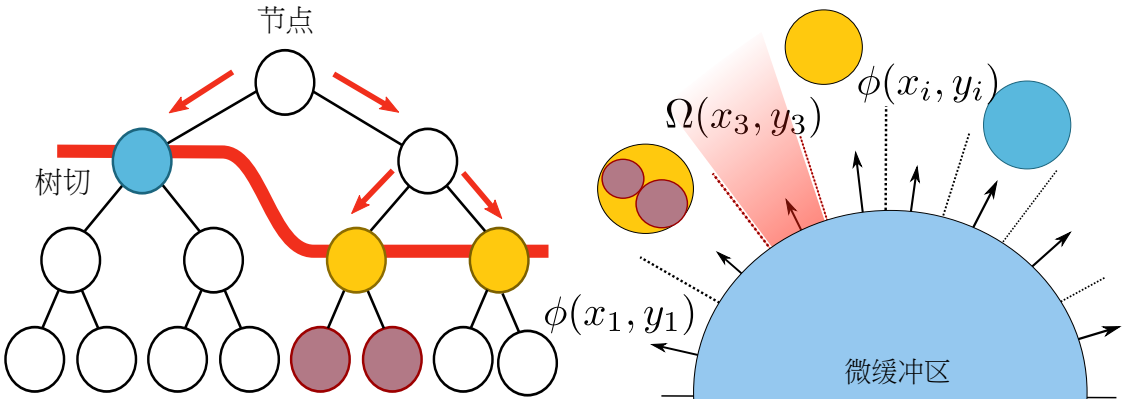


图 2.5 渲染阶段. 遍历点云层次结构找到树切 (左图)，然后将树切中的节点向着色点的微缓冲进行投影 (右图)。 微缓冲区的每个像素 (x_i, y_i) 对应了方向 $\Phi(x_i, y_i)$, 并且产生了立体角 $\Omega(x_i, y_i)$。 当节点的立体角不超过一个像素时则进行投影 (如图中蓝色节点和两个黄色节点)。

$\Phi(x, y) = (x, y, \sqrt{1 - x^2 - y^2})$，也可以使用自定义的映射。 立方体微缓冲区的优点是，在计算环境映射时，每个着色点环境贴图在立方体微缓冲区上的投影都是相同的； 缺点是，微缓冲区有 6 个面，所以投影计算比较复杂，而且每个像素的立体角是不同的。 半球微缓冲区在使用标准半球映射时，每个像素的立体角相同。

将树切中的节点向该微缓冲区投影，如果节点距离着色点很近，那么该节点中的点向微缓冲区进行投影， 在有重叠的情况下，则使用光线投射 (ray casting) 来计算可见性。

当遍历和投影全部结束时，将微缓冲区中每个像素的颜色值与着色点对应方向的 BRDF 值以及立体角进行卷积得到着色点最终的间接光照值 (出射辐射亮度)。位于 x、法向为 n 的着色点在 ω_o 方向上的出射辐射亮度计算公式如下：

$$L(x, \omega_o) = \sum_{k=0}^{M} L_i(k)\rho(x, \omega_k, \omega_o)s(k)(\omega_k \cdot n), \tag{2.1}$$

其中 M 是微缓冲区像素的个数，ω_k 是对应于微缓冲区的像素 k 的出射方向， $L_i(k)$ 是像素 k 中的颜色值， $s(k)$ 是像素 k 的立体角。

2.3 PBGI 的其他应用

除了模拟色溢和环境光遮挡外，PBGI 还可以应用到以下情况：光泽反射 (glossy reflection)、面积光、多次反射 (multi-bounces) 等。 在光泽反射中，只需要使用某个角度内的微缓冲区 (根据 BRDF 确定)，在该角度以外的节点不需要遍历和投影。 在处理面积光时，产生点云阶段也要将面积光进行采样，并且也和其他的点一起组织到点云层次结构中， 将点云树遍历和投影，即可计算出面积光产生的直接光照。 另外，PBGI 还可以计算扩展到多次反射的情况， 将点云中的每个点当作着色点，计算出它们的间接光照，此时点云中缓存了直接光照和一次间接光照， 当使用该点云产生的点云层次结构对着色点进行间接光照计算时，所得的结果即为一次间接光照和二次间接光照的和，依次类推下去。

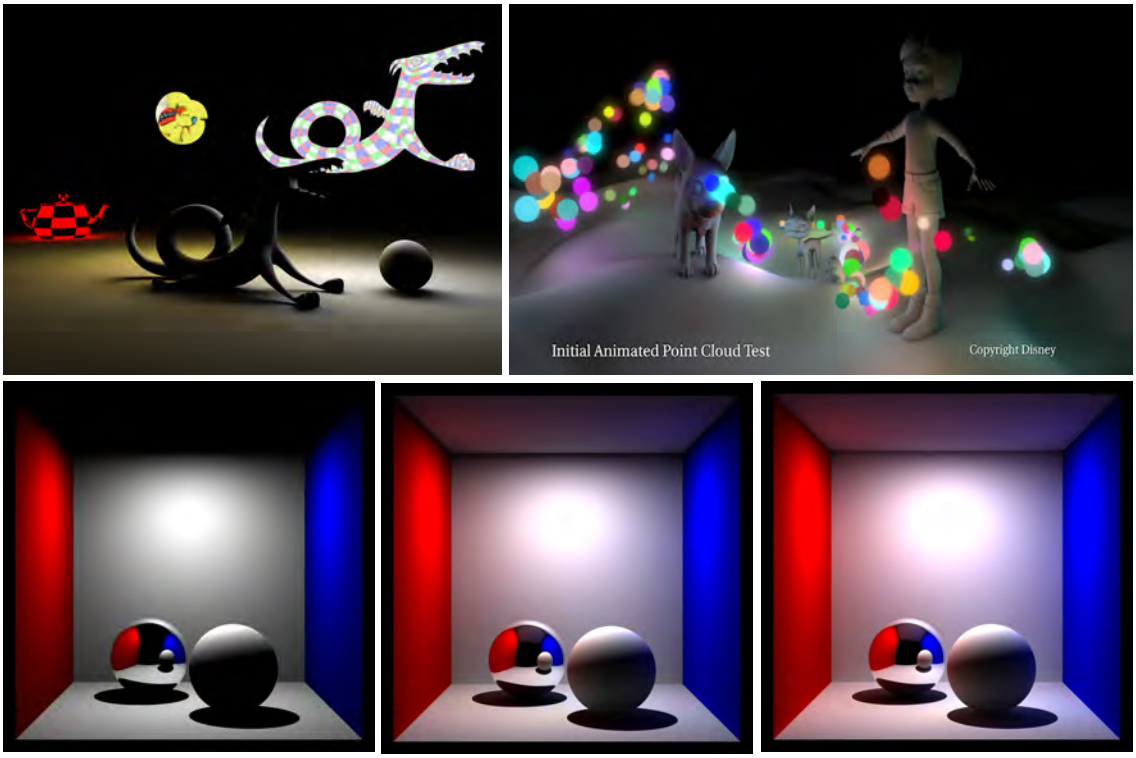


图 2.6 上面两张图是 PBGI 应用在面积光中的效果； 下面三张图分别表示直接光照、一次反射和二次反射的效果， 图片来源于 [24]。

2.4 分析

优点 PBGI 算法主要特点在于： 第一，使用了点云层次结构来组织点云，从而可以充分使用 LoD 技术，在保证质量的前提下有较高效率； 第二，点云数据中不仅包含了光照信息也包含了几何信息，从而使得计算间接阴影比较容易； 第三，提出了微缓冲区 (相对于光线跟踪而言，微缓冲区具有更高效率) 技术来计算间接阴影（或者说节点的可见性判断）。 以上三点决定了 PGBI 的优势:相比于光线跟踪算法可以更高效地计算色溢等效果，并且渲染结果是无噪声的。

缺点 相比光线跟踪等算法，PGBI 算法主要有以下缺点： 第一，算法是有偏的(biased)，甚至不具有一致性(consistence)； 第二，需要占用大量的内存； 第三，不支持焦散等非漫反射光线传递的模拟。

基于 PBGI 的优缺点，我们在下面的章节中展开研究。

第 三 章　基于分解的点缓存全局光照

PBGI 算法包含两个阶段，预处理阶段和渲染阶段。渲染阶段主要包含三个步骤：遍历点云树找到树切(tree-cut)、向微缓冲区投影以及与 BRDF 进行卷积。 其中，遍历和投影过程在整个渲染时间中占很大比例。 由于空间连贯性，两个空间相近的着色点具有类似的树切，因此遍历过程中存在冗余，如图 3.1 所示。

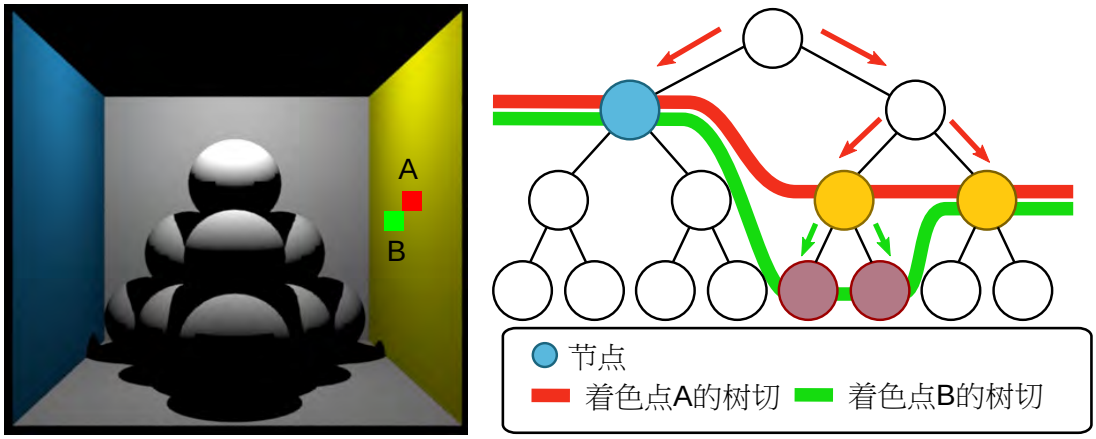


图 3.1 左图中是场景中两个相近的着色点 A 和 B，右图中是它们的树切。 A 的树切需要 4 步遍历操作， B 的树切需要 6 步遍历操作，而其中的 4 步与 A 中相同。

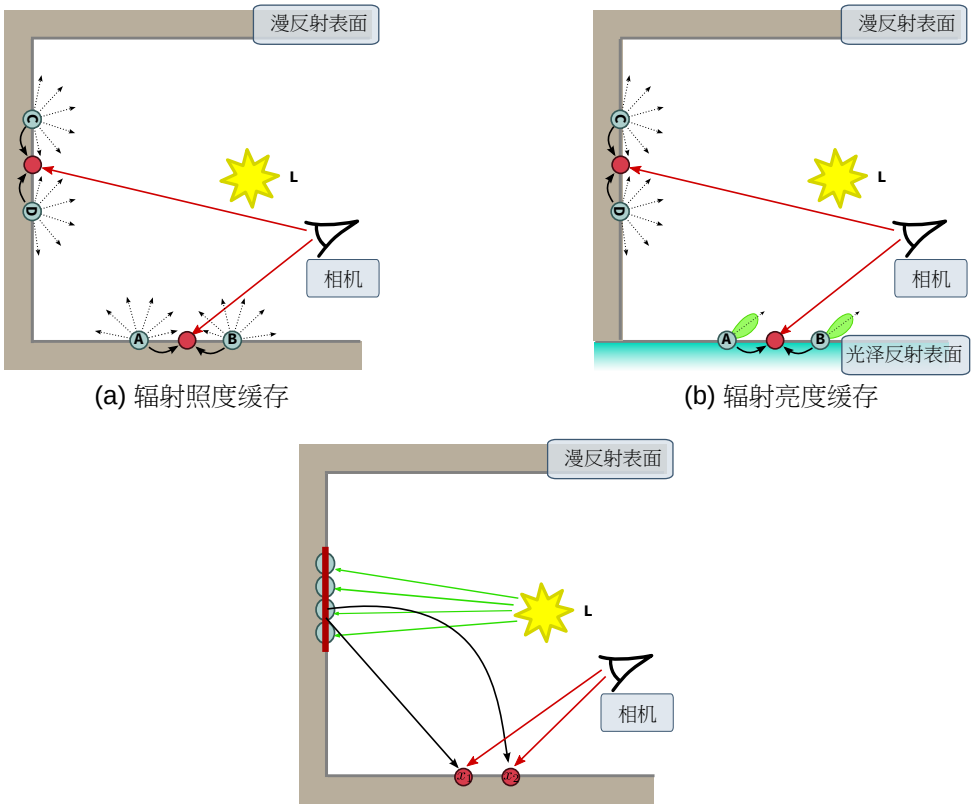
本章针对遍历和投影中存在的冗余问题，利用聚类方法得到相似的着色点，继而提出基于分解的方法来重用聚类中着色点之间的树切，并且进一步重用微缓冲区， 在保证渲染质量的前提下来提高渲染效率。 该方法可以应用在已有的 PBGI 框架中，易于实现。 本方法的适用范围同 PBGI 一致，但是在平滑的场景中具有更大优势。 此外，我们的方法具有与 PBGI 相同的时间连贯性，即将该算法应用于渲染动画时，在帧与帧之间不存在闪烁。 本章中首先介绍与树切、重用技术等相关的工作，然后着重提出基于分解的点缓存全局光照算法。

3.1 相关工作

3.1.1 PBGI 中关于树切的研究

Maletz 等人在 [35] 中首次提出树切的定义，并且使用重要性驱动来选择树切。Holländer 等人 [36] 实现了基于 GPU 的 PBGI，达到了实时的效率，他们提出在每

帧之间使用 Lazy 模式，即从时间连贯性上来重用树切。 受到该文献的启发，我们提出在空间连贯性上进行重用。 Tabellion [37] 中提出了高动态范围图像(High Dynamic Range Imaging 简写为 HDR)作为环境映射情况下的树切选择算法， 在 HDR 中亮度很高的像素处，使用较小的立体角，即遍历到点云层次结构更深的部分。



(a) 辐射照度缓存　(b) 辐射亮度缓存

(c) 树切缓存

图 3.2 辐射照度缓存: 首先将一些采样点分布到场景的可见表面(漫反射)上。 对每个采样点(A, B, C, D)，在它法向方向的半球上进行渲染(比如利用光线跟踪进行最终聚集)，计算出这些采样点的辐射照度(RGB)。 在计算场景中着色点(红色圆圈)的全局光照时，通过对周围采样点的辐射照度插值得到。 如果该着色点周围没有采样点，则直接计算该着色点的全局光照并且加入到采样点列表中。 辐射亮度缓存: 将一些采样点放在场景的可见表面(不要求漫反射表面)上， 然后在半球内计算各个方向上的入射辐射亮度，并且使用球谐函数系数表示。 每个着色点通过周围采样点来插值得到其入射辐射亮度的系数，并且与表示 BRDF 的系数相乘之后得到该着色点的出射辐射亮度。 树切缓存: 对一些着色点(x_1，x_2)，共享部分树切(蓝色节点组成的红色树切)。

3.1.2 渲染中的重用技术

利用连贯性进行重用是全局光照算法中的一项重要技术。通过重用技术可以减少重复性计算，从而有效地提高效率。 重用技术常用于辐射照度、辐射亮度、树切等的计算，如图 3.2 所示。 Ward 等人在 [38] 提出了只计算部分像素的辐射照度，其他的像素则通过对已计算像素的辐射照度进行插值得到。 [39] 中利用梯度信息进行插值，从而得到平滑的效果，即辐射照度缓存 (Irradiance Caching)。 同样地，Wang 等人 [40] 使用 k-means 算法对着色点进行聚类，并且对每个聚类只计算一个点的辐射照度(使用光子映射)， 而其他点则通过插值产生，从而使得该方法达到可交互效率。 Wang 等人在 [41] 中也通过聚类进行辐射照度的重用，只计算聚类中心着色点的间接光照值， 聚类内的其他着色点均使用该值， 通过判断与周围聚类的误差来判断是否要缩小聚类，最终满足预设结果。 辐射亮度缓存 (Radiance Caching) 技术是 Krivanek 在 [42] 中提出的， 使用球谐函数 (SH) 系数存储采样点的入射辐射亮度，从而缓存方向性信息， 将缓存的采样点的辐射亮度系数插值得到着色点的辐射亮度系数，最后与表示 BRDF 的 SH 系数(已经包含了 cosine 项)相乘后得到最终间接光照值， 此方法克服了之前的辐射照度重用中只适用于漫反射表面的缺陷。 Holländer 在 [36] 中提出在帧与帧之间重用树切。

3.1.3 远-近划分

在一些文献中，通过远近的划分来提高效率。 Shanmugam 等人在 [43] 中依据对着色点的远近距离对树切缓存进划分，应用于环境光遮挡中。 Arikan 等人在 [44] 中将远近划分应用于最终聚集 (final gathering)。 在本章节中，我们也对树切中的节点进行远近划分，远节点直接被多个着色点共用，近节点被进一步遍历。

3.2 算法概述

我们利用着色点的空间连贯性对树切和投影进行分解，提出了基于分解的点缓存全局光照算法 (Factorized Point Based Global Illumination，简称为 FPBGI)， 主要分为以下三部分(如图 3.3):

1. 对于每个块内的着色点按照相似性进行聚类，并且在每个聚类中选择一个活跃着色点(*active* receiver);

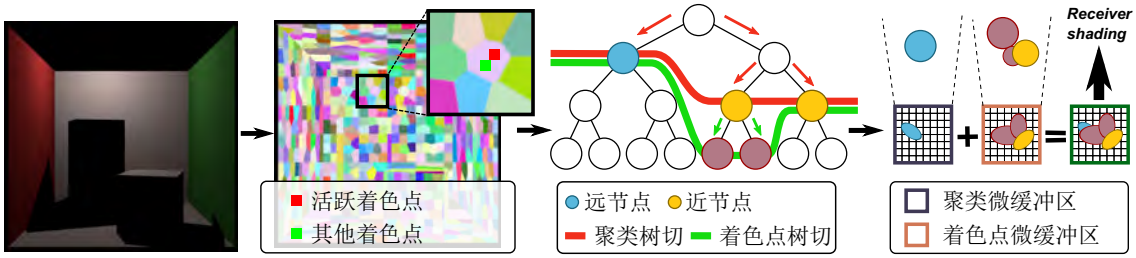


图 3.3 流程. 在左图中，计算出每个分块的着色点，并且根据着色点之间的相似性对一个块中的着色点进行聚类(左中图)。 对一个给定的聚类，计算聚类树切(右中图，红色)，对聚类树切中的近节点，每个着色点单独遍历从而更新他们的各自的树切(右中图，绿色)。将聚类树切中的远节点投影在聚类微缓冲区中(右图，紫色)，而每个着色点单独更新的节点投影在着色点微缓冲区上(右图，橙色)。 对每个着色点，将着色点微缓冲区与聚类微缓冲区融合，并且最终与 BRDF 卷积后得到该着色点的间接光照。

2. 对于每个聚类，从活跃着色点计算出了粗糙的树切称为聚类树切(cluster cut)，并且将该树切中的远节点投影到聚类微缓冲区上;
3. 对于每个着色点，对树切中较近的节点进行遍历，并且将更新的节点投影到着色点微缓冲区中， 最后着色点微缓冲区与聚类微缓冲区进行融合，并且与 BRDF 进行卷积后得到着色点的间接光照。

此算法可以减少相邻的着色点之间遍历和投影操作，从而使得渲染速度提高 2 到 4 倍。

3.3 基于分解的点缓存全局光照

3.3.1 术语介绍

在点云层次结构中，将节点表示为 n_i，定义路径为从根节点到叶节点所有节点形成的集合，表示为 P，树切定义为对于点云层次结构中的每个路径中有且只有一个节点所形成的节点集合，表示为 C，如图 3.4 所示。我们定义节点间的大小关系如下：如果节点 n_i 是节点 n_j 的父节点，那么 $n_i > n_j$ ，或者 $n_j < n_i$。如果节点 n_i 和节点 n_j 是同一个节点，那么 $n_i = n_j$ 。 通过该定义得知，只有在同一条路径上的节点才存在大小关系。 该关系具有传递性，即如果 $n_i > n_j$ 并且 $n_j > n_k$，那么 $n_i > n_k$。 如果两个节点存在大小关系，那么可以对这两个节点进行差运算，表

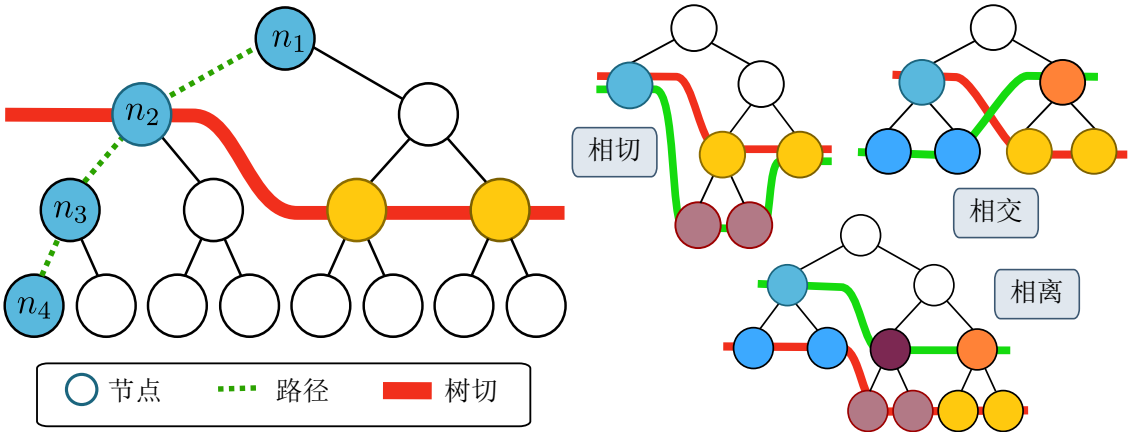


图 3.4 树切与路径。绿色虚线所经过的节点构成的集合称为路径；红色实线所经过的节点构成的集合称为树切。根据节点间的距离定义可以得到 $n_2-n_1=1$, $n_4-n_2=2$, $n_4-n_1=3$。

示两个节点在路径上的距离。若节点 n_i 是节点 n_j 的父节点，那么 $n_i-n_j=1$；同理，若节点 n_j 是节点 n_k 的父节点，则有 $n_j-n_k=1$，$n_i-n_k=2$，以次类推。

令 C_1 和 C_2 表示两个树切，$C_1=\{n_i\}, C_2=\{n_j\}$，两个树切之间的距离为：$d(C_1,C_2)=\sum_{n_i\in C_1}\|n_i-n_k'\|$. 其中 $n_k'\in C_2$，并且与 n_i 存在大小关系。此外，我们定义两个树切的三种关系：相离，相切和相交，如图 3.4 所示。如果 C_2 中的所有节点，在 C_1 中存在大于该节点的节点，那么我们称 $C_2\prec C_1$，形式化表示如下：

$$if\ \ \forall n_j\in C_2,\ \ \exists n_i\in C_1, s.t.\ \ n_j>n_i,\ \ then\ \ C_2\prec C_1.$$

如果 $C_2\prec C_1$，那么这两个树切一定不是相交的。因为如果两者相交，即在 C_1 中存在某节点小于 C_2 中某个节点(设为 n_j)，根据 $C_2\prec C_1$，在 C_1 中一定存在一个节点大于 n_j，因此在 C_1 中既存在大于 n_j 的节点又存在小于 n_j 的节点，这与树切的定义矛盾(在每个路径上有且只能有一个节点)，因此推理成立。

3.3.2 着色点聚类

在 PBGI 算法中，对点云层次结构进行遍历来找到适合的树切。当前节点是否直接投影或者需要进一步遍历到其子节点由当前节点与着色点产生的立体角决定，如图 3.5 所示。立体角的计算公式是

$$\delta=\frac{A}{d^2}.$$

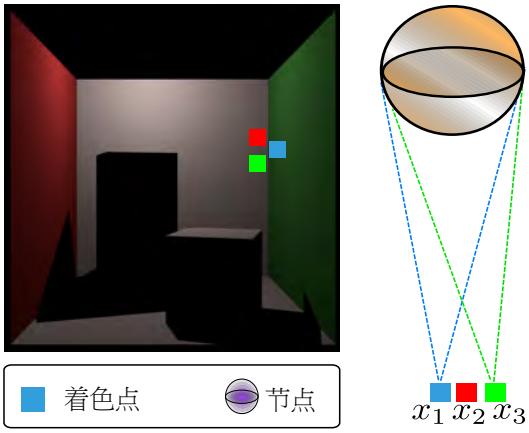


图 3.5 立体角。 三个着色点 x_1，x_2 和 x_3 分别如左图所示， 右图中表示了某节点分别对三个着色点的立体角。

其中，A 表示节点的投影面积，d 表示节点与当前着色点之间的距离。 立体角是决定树切中节点的最重要指标。 一般来说，两个位置相近的着色点(当两个着色点的距离远远小于节点与着色点之间的距离时)，他们的立体角也相近。 此外，着色点的法向也至关重要，直接决定了节点是否在着色点的朝向范围内。 因此，本文做出假设，位置和法向相近的着色点有相近的树切。 我们提出度量 $\mathcal{D}$ 表示位置/法向相似度。

考虑到并行性因素(每个线程负责一个块)，首先将图像分块，再对每个块中的着色点使用 k-means 算法的变形进行聚类，算法如下。

1. 在一个块中，随机选取 k 个着色点作为初始中心点；
2. 将块中的着色点分配给距离它最近的中心点，其中，“最近”的含义按照度量 $\mathcal{D}$ 来确定；
3. 更新每个聚类的中心，并且重新开始 2。

迭代达到预设次数后，计算距离最终中心点最近的着色点，将该着色点称为该聚类的活跃着色点 $\mathbf{x}_C$。

$\mathcal{D}$ 的定义有多种，根据 Cohen-Steiner 等 [45] 提出的方法，将 $\mathcal{D}$ 定义如下：

$$\mathcal{D}(\mathbf{x}, \mathbf{c}) = \|\mathbf{p_c} - \mathbf{p_x}\|^2 + \alpha\|\mathbf{n_c} - \mathbf{n_x}\|^2.$$

其中 $\mathbf{x}$ 表示着色点，$\mathbf{c}$ 表示一个聚类 C 的中心点，$\mathbf{p}$ 和 $\mathbf{n}$ 分别表示它们在 $\mathbb{R}^3$ 中的位置和法向，α 表示权重，不失一般性，将该值设置为该块内着色点包围盒的对角线长度。每个中心点 $\mathbf{c}$ 的位置和法向更新的公式如下：

$$\mathbf{p_c} = \frac{\sum_{\mathbf{x}\in C} \mathbf{p}_x}{card(C)} \quad \mathbf{n_c} = \frac{\sum_{\mathbf{x}\in C} \mathbf{n_x}}{\| \sum_{\mathbf{x}\in C} \mathbf{n_x} \|}.$$

其中 $card(C)$ 表示 C 中元素的个数。图 3.3 的第 2 个图展示了 Cornell Box 聚类后的结果。

3.3.3 聚类的树切和微缓冲区

我们可以为聚类中的每个着色点找到一个树切，根据上节中的假设，这些树切是相似的。我们的目标是找到大于该聚类中的所有着色点树切并且与它们距离最小的树切，称之为聚类树切 C，表示为

$$min \sum_{i=1}^{N} d(C, C_i).$$

其中 N 表示聚类中着色点的个数，C_i 表示着色点 i 对应的树切。

每个着色点的树切是未知的，计算聚类树切的目的就是简化计算着色点树切的过程，所以无法从聚类树切的定义来计算它。我们通过该聚类的活跃着色点计算得到 C 的近似解。对活跃着色点遍历点云层次结构，从根部开始，但是与 PBGI 相比，采用更大的立体角阈值，即获得粗糙的树切，用于近似聚类树切。

令聚类 C 的半径为 r，任意节点 n_i 与 $\mathbf{x}_C$ 的距离为 d，我们定义 γ 为：$\gamma = \frac{r}{d}$。如果 $\gamma > \epsilon$，则节点 n_i 称为远节点，反之称为近节点，如图 3.6 所示。

远节点直接被聚类中的着色点所共享，不再进行遍历；对近节点，由于它们对聚类中的不同着色点的贡献差异可能较大，因此需要每个着色点分别进行进一步遍历。每个着色点使用微缓冲区来计算找到的树切中节点的可见性，同样的，我们使用聚类微缓冲区来计算聚类树切中远节点的可见性。该微缓冲区根据活跃着色点的参数构建，微缓冲区可以采用全局坐标系或者活跃着色点的局部坐标系。构建完成后，将聚类树切中的远节点向聚类微缓冲区进行投影。

3.3.4 着色点树切，微缓冲区和着色

每个着色点，对聚类树切中的近节点进行处理，即从该近节点开始遍历，直到对于每个着色点满足立体角的条件为止。与 PBGI 算法不同的是，FPBGI 从近

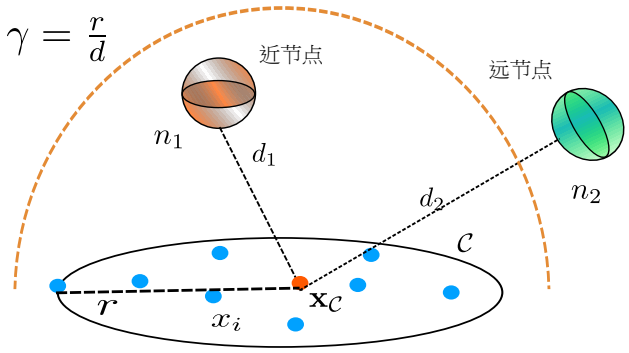


图 3.6 远节点和近节点。 黑色椭圆表示了聚类 C，半径为 r，其中的点表示了该聚类中的着色点， 红色的点表示活跃着色点 $\mathbf{x}_C$。 通过计算聚类半径与距离的比值来判断两个节点 (n_1 和 n_2，与活跃着色点的距离分别为 d_1 和 d_2) 为远节点或者近节点。

表 3.1 场景性能和误差统计.

Scene	**Points**	**Total Time**		**Error**	
		Full Rendering			
		PBGI	*FPBGI*	*PDP*	*MSE*
CornellBox	88.88K	5.72m	2.60m	103	8.79e-6
Bunny	1.00M	4.49m	2.03m	61	1.31e-4
ItalianCity	8.38M	5.52m	2.34m	235	8.15e-5
Sponza	16.19M	26.72m	7.04m	15	2.51e-5

节点开始遍历而不是从点云层次结构的根节点开始遍历。 在该遍历过程中增加的并且最后形成新树切的节点标记为 *refined*。 最后，我们将近节点和标记为 *refined* 的节点在该着色点的微缓冲区上进行投影。

此时，在聚类微缓冲区中存储了距离该聚类较远的节点产生的入射辐射亮度，而在着色点微缓冲区中存储了较近节点产生的入射辐射亮度。 将着色点微缓冲区与聚类微缓冲区按照所存储的深度信息进行融合，得到着色点完整的微缓冲区。在进行微缓冲区融合时，需要注意微缓冲区坐标系的一致性，如果微缓冲区采用全局坐标系，那么可以直接计算； 如果使用的局部坐标系，则需要先将着色点微缓冲区和聚类微缓冲区变换到相同的坐标系中。

最终，完整的微缓冲区与着色点的 BRDF 以及 cosine 等进行卷积，得到着色点的间接光照值。

表 3.2 各阶段渲染时间统计.

Clustering	**Cut Computation**		**Micro-Rasterization**	
Time(s)	*PBGI*	*FPBGI*	*PBGI*	*FPBGI*
0.68s	2.46m	0.65m	1.73m	1.19m
0.87s	2.38m	0.56m	1.55m	1.08m
0.87s	3.65m	1.08m	1.23m	0.64m
0.87s	19.71m	3.77m	5.40m	1.68m

3.4 结果及讨论

我们的技术在 Mitsuba Renderer [2] 上进行实现，使用泊松盘采样算法产生点云，并且同 PBGI 算法进行了对照。 实验的硬件环境是 2.67 GHz Intel i7 (8核)处理器，9 GB 内存。 所有的渲染结果计算了一次反弹的间接光照，分辨率为 1280×1000 (Cornel Box 场景除外，使用 1024×1024 分辨率)，并且使用 32×32 分块。 在所有的对照中，使用均方差 (MSE) 和像素感知差 (PDP) [46] 来衡量误差。在渲染过程中，使用 PBGI 技术计算间接光照，而直接光照则采用光线跟踪进行计算。

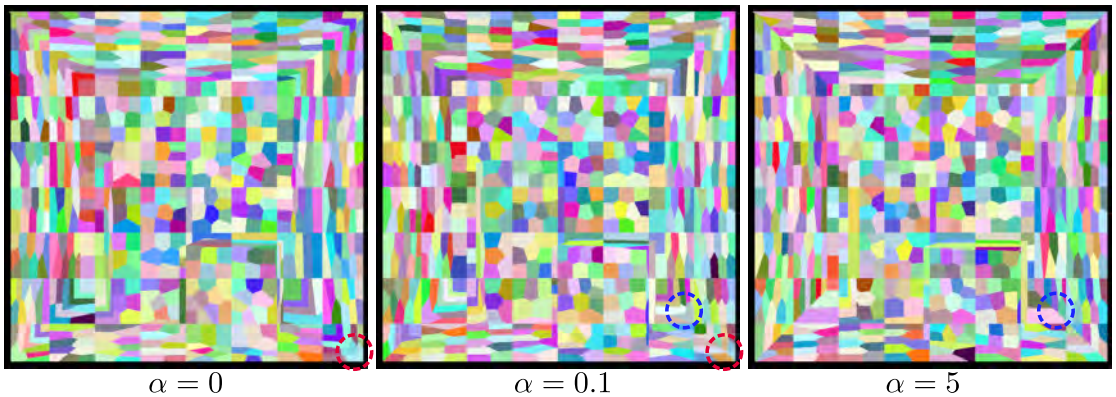


图 3.7 Cornel Box 场景进行 *k*-means，随着 α 变化的结果对比。 图中标注的虚线圆圈标识出了由于参数 α 的不同，导致法向所起作用的不同。

在图 3.7 中，我们给出了三种不同参数下的聚类结果。

在图 3.8、图 3.9 以及图 3.10 中， 我们通过三个场景将 FPBGI 与 PBGI 算法进行了对比。 从整体来看，我们算法的误差从数值角度(MSE)和感知角度(PDP)都是

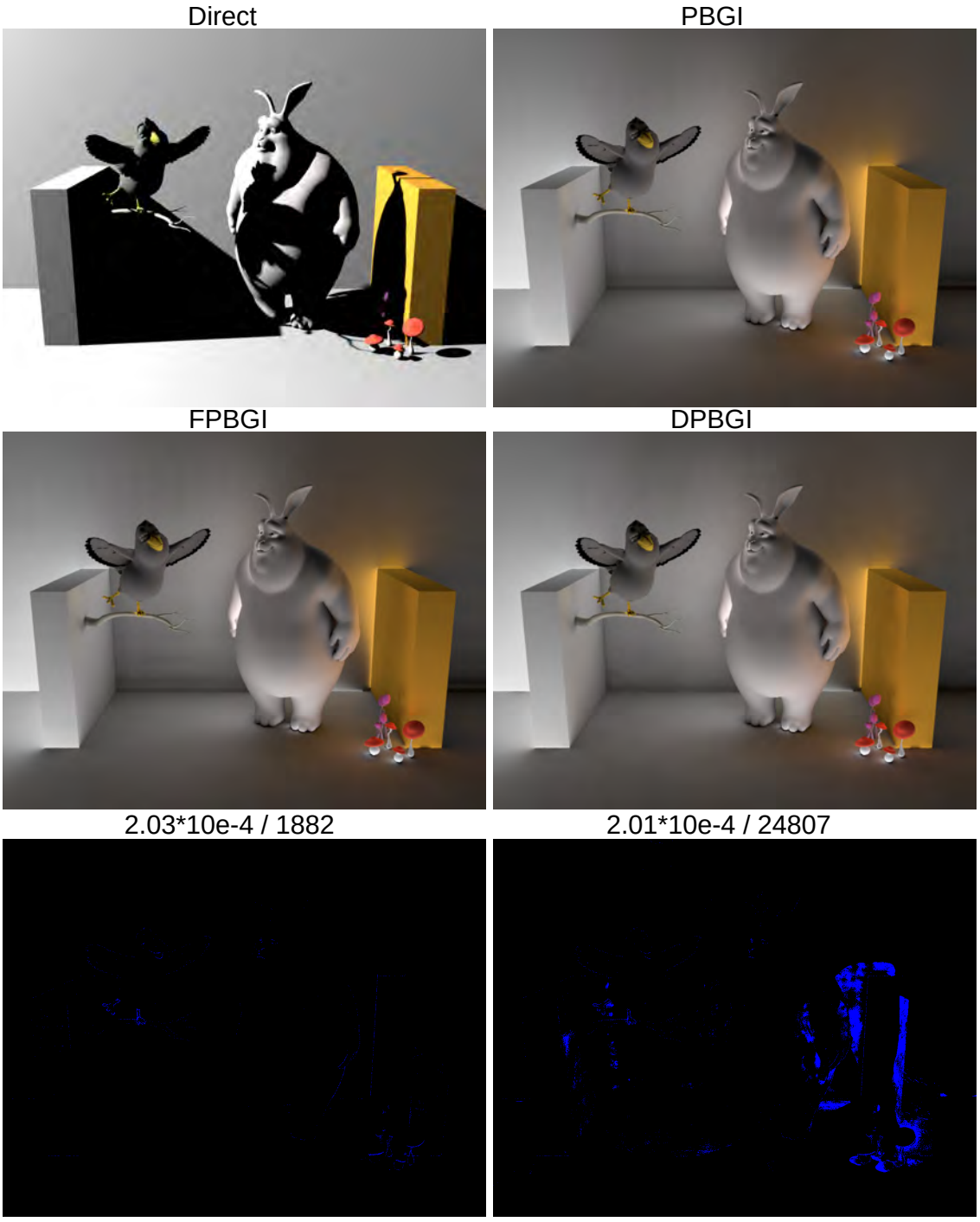


图 3.8 场景 Bunny: FPBGI、DPBGI 相对于 PBGI 的误差分析。 第一行的左图为直接光照的结果，右图为在 PBGI 计算得到的间接光照的结果， 第二行的左图为 FPBGI 计算得到的间接光照结果，右图为 DPBGI 计算得到的间接光照结果， 第三行为误差图片，感知误差 [46] 表示为：黑色为无误差，蓝色为有误差。 在误差图上面的两个数字分别为两个 RGB 图像之间的 MSE 误差和感知误差。

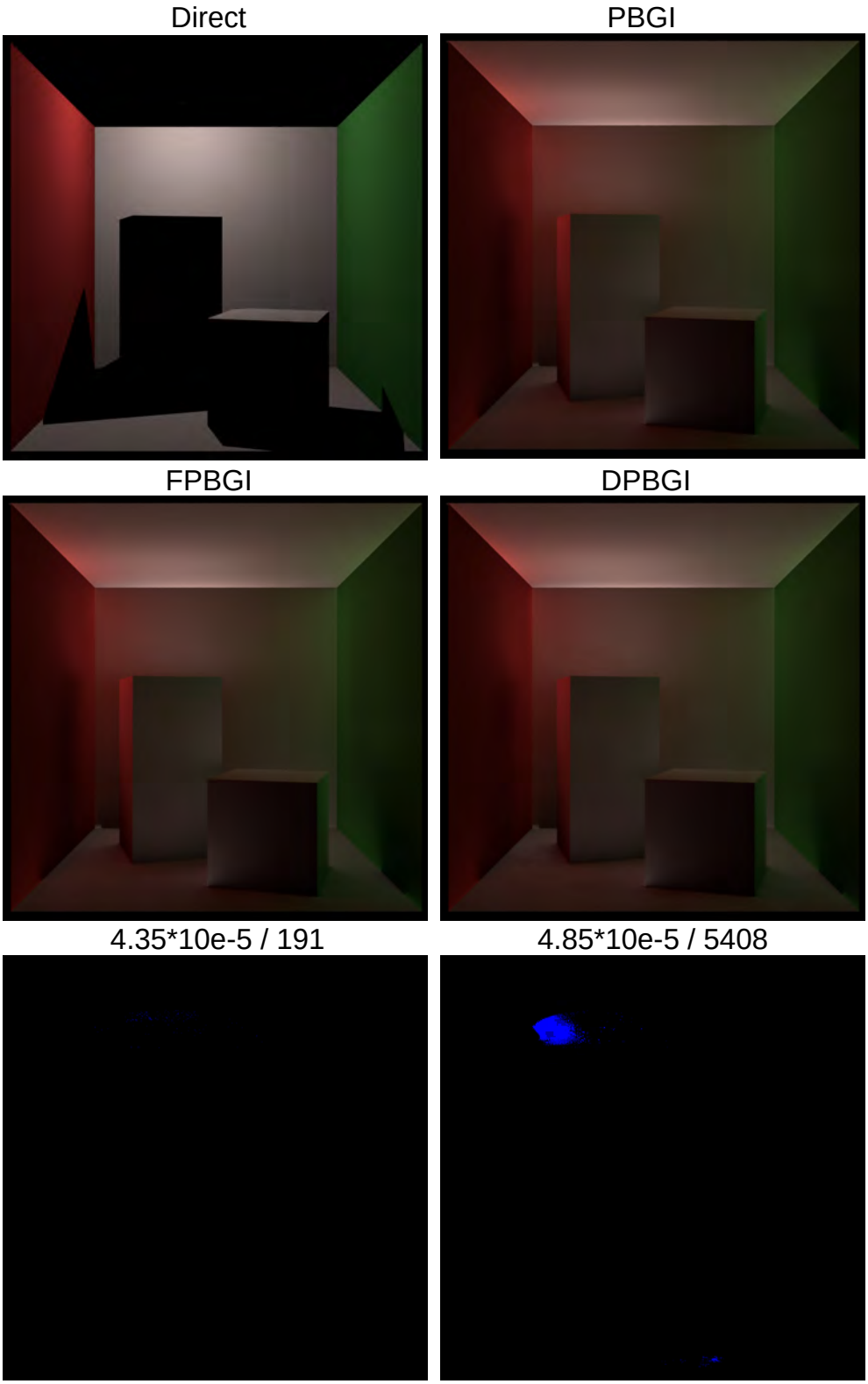


图 3.9 场景 Cornel Box： FPBGI、DPBGI 相对于 PBGI 的误差分析。

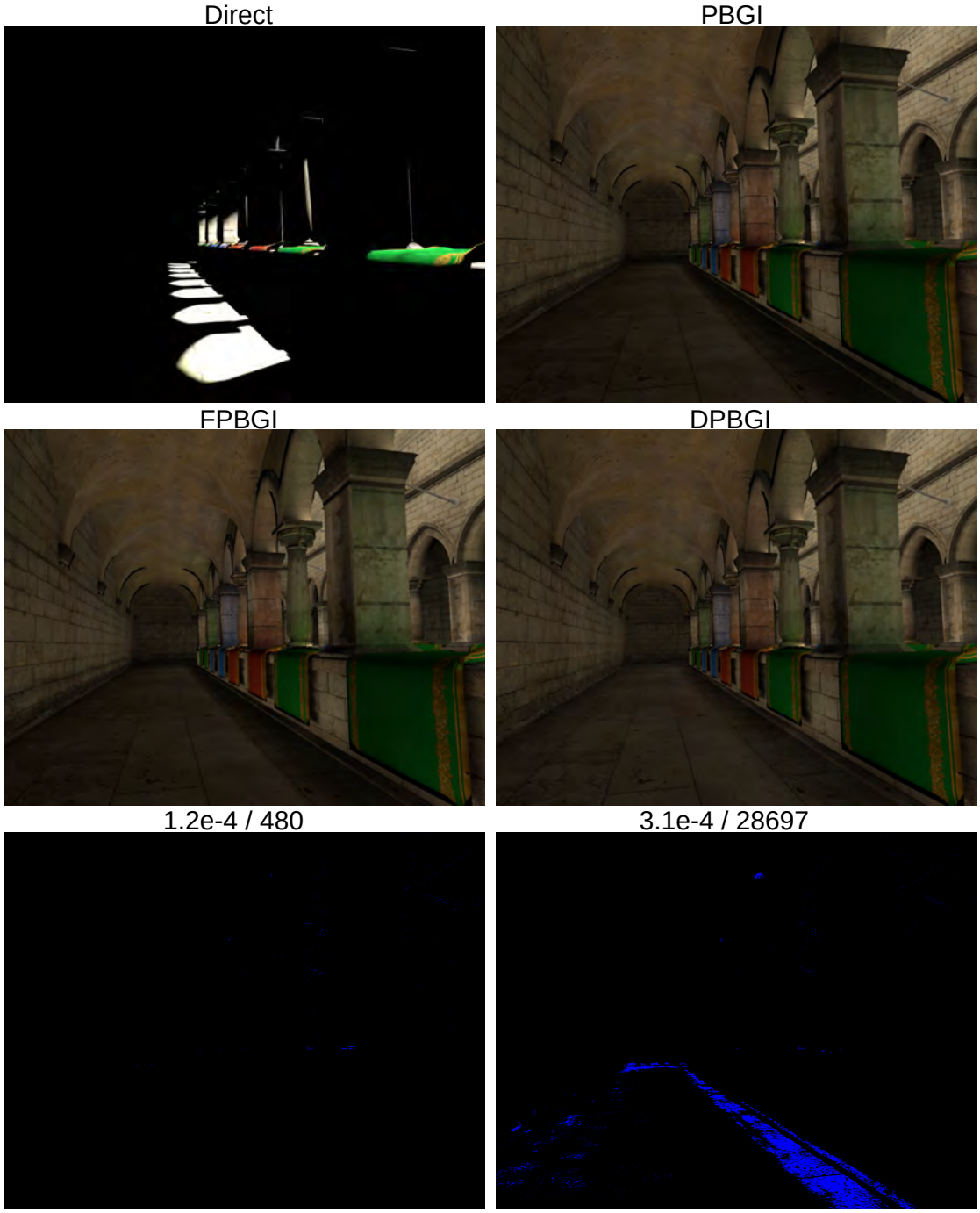


图 3.10 场景 Sponza： FPBGI、DPBGI 相对于 PBGI 的误差分析。 第一行的左图为直接光照的结果，右图为在 PBGI 计算得到的间接光照的结果， 第二行的左图为 FPBGI 计算得到的间接光照结果，右图为 DPBGI 计算得到的间接光照结果， 第三行为误差图片，感知误差 [46] 表示为：黑色为无误差，蓝色为有误差。 在误差图上面的两个数字分别为两个 RGB 图像之间的 MSE 误差和感知误差。

可忽略的。PBGI 算法可以通过降低微缓冲区的分辨率或者较高的立体角来获得速度上的提升， 因此我们也将 FPBGI 算法与 *degraded* PBGI (DPBGI) 进行了对比。DPGBI 的含义是降低微缓冲区的分辨率提升 PBGI 的速度使之与 FPBGI 有相似的渲染时间。 在这种情况下，我们发现 DPGBI 有较大的误差，并且有明显的走样出现。

表 3.1 和表 3.2 中统计了图 3.11 和图 3.12 中的几个场景的渲染时间和误差情况。 表 3.1 统计了总渲染时间，该渲染时间指的是总渲染时间，除了树切计算和投影外还包括了点云的产生，点云层次结构的构建以及最后的 BRDF 卷积时间。 与 PBGI 算法相比，FPBGI 可以获得 2.2 到 3.8 的加速比。 表 3.2 统计了每个分步骤的时间，包括了聚类、树切计算以及向微缓冲区投影的时间。 通过统计，FPBGI 相比 PBGI 在树切计算部分可以获得 3.4 倍到 5.2 倍的加速比； 在投影部分可以获得 1.4 倍到 3.2 倍的加速比，并且聚类时间是可以忽略的。

在图 3.11 和图 3.12 中， 我们给出了最终渲染效果(直接光照+间接光照)在路径跟踪(PTS)、 PBGI 和 FPBGI 之间的视觉上的对比。 通过对比，我们发现 FPBGI 同 PBGI 一样得到与 PTS 的相近的结果，但是速度上却比 PBGI 算法快数倍。

我们对 FPBGI 算法中的两个主要参数进行了分析，它们分别是每块聚类的个数 k 和远近节点判断的阈值 ϵ。 图 4 中展示了在不同参数下，场景 Sponza 的渲染结果。 我们发现，在 k 值固定时，随着 ϵ 的增加，图像在数值和感知上的误差都在增大， 这是因为越来越多的节点被确定为远节点，即更多的节点被重用。 当 ϵ 固定时，随着 k 的增加，图像在数值上和感知上的误差都在减小， 这是因为随着聚类数的增加，而每个块中着色点数是确定的，因此每个聚类在变小，即聚类的半径在变小， 根据远近节点的判断准则，此时较少的节点被确定为远节点，因此误差比较小。 当 ϵ 值比较大，而 k 值比较小时，会有明显的走样。 图 5 表示了场景渲染时间随着这两个参数变化而产生的变化。 正如我们上面分析的，越多的节点被确定为远节点时，越多的节点会出现重用，因此渲染时间更短，即与 PBGI 相比具有更高的加速比。 在实验过程中，$k = 100$ 和 $\epsilon = 10e^{-2}$ 对于实验的所有场景都是适用的。 另外，为了证明算法具有时间连贯性(即在帧与帧之间没有闪烁)，我们给出了几段视频，分别包含了运动的光源、相机和模型。

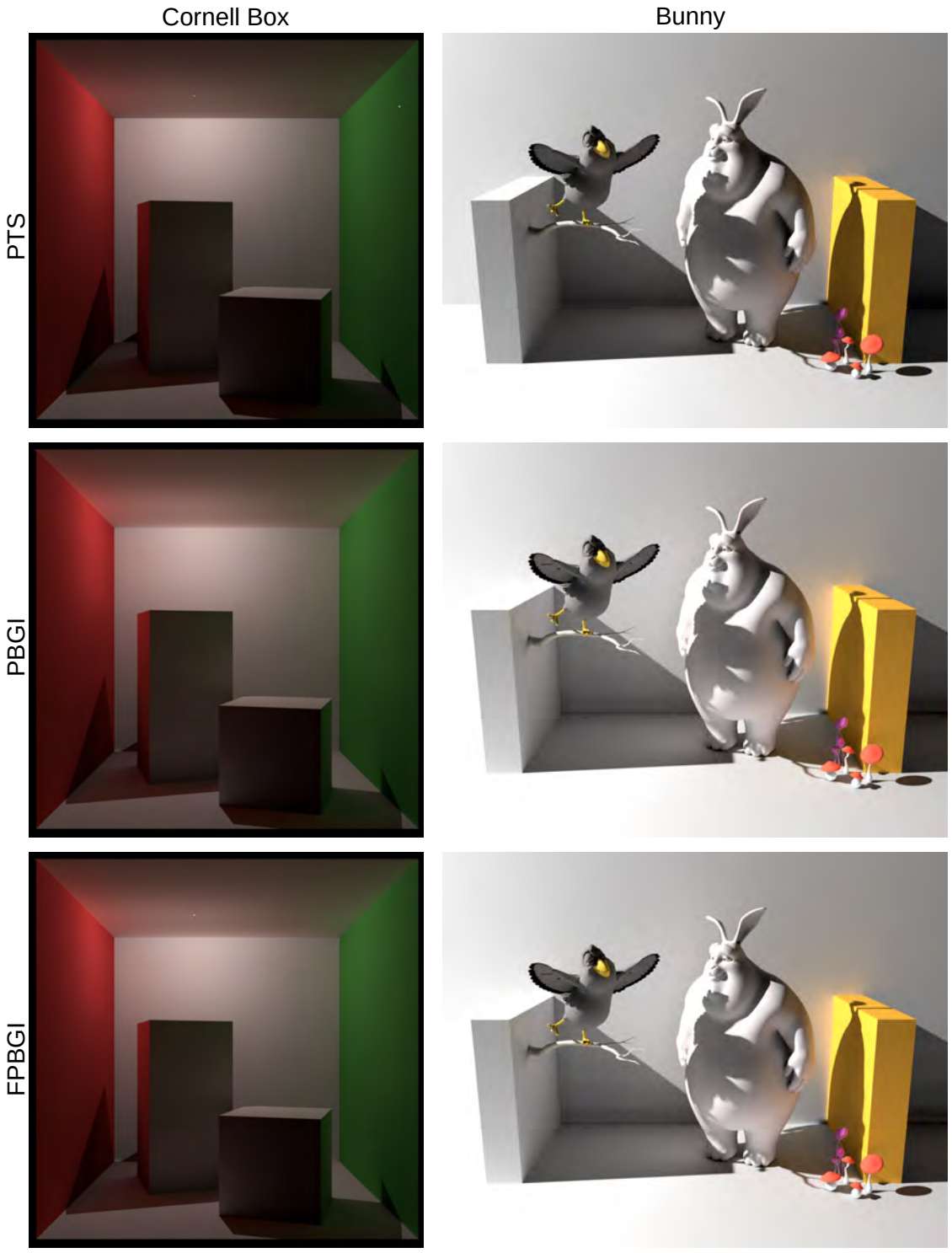


图 3.11 场景 Cornel Box 和场景 Bunny 最终渲染结果在 PTS，PBGI 和 FPBGI 间的视觉对比，图中的结果包括了直接光照和一次间接光照。

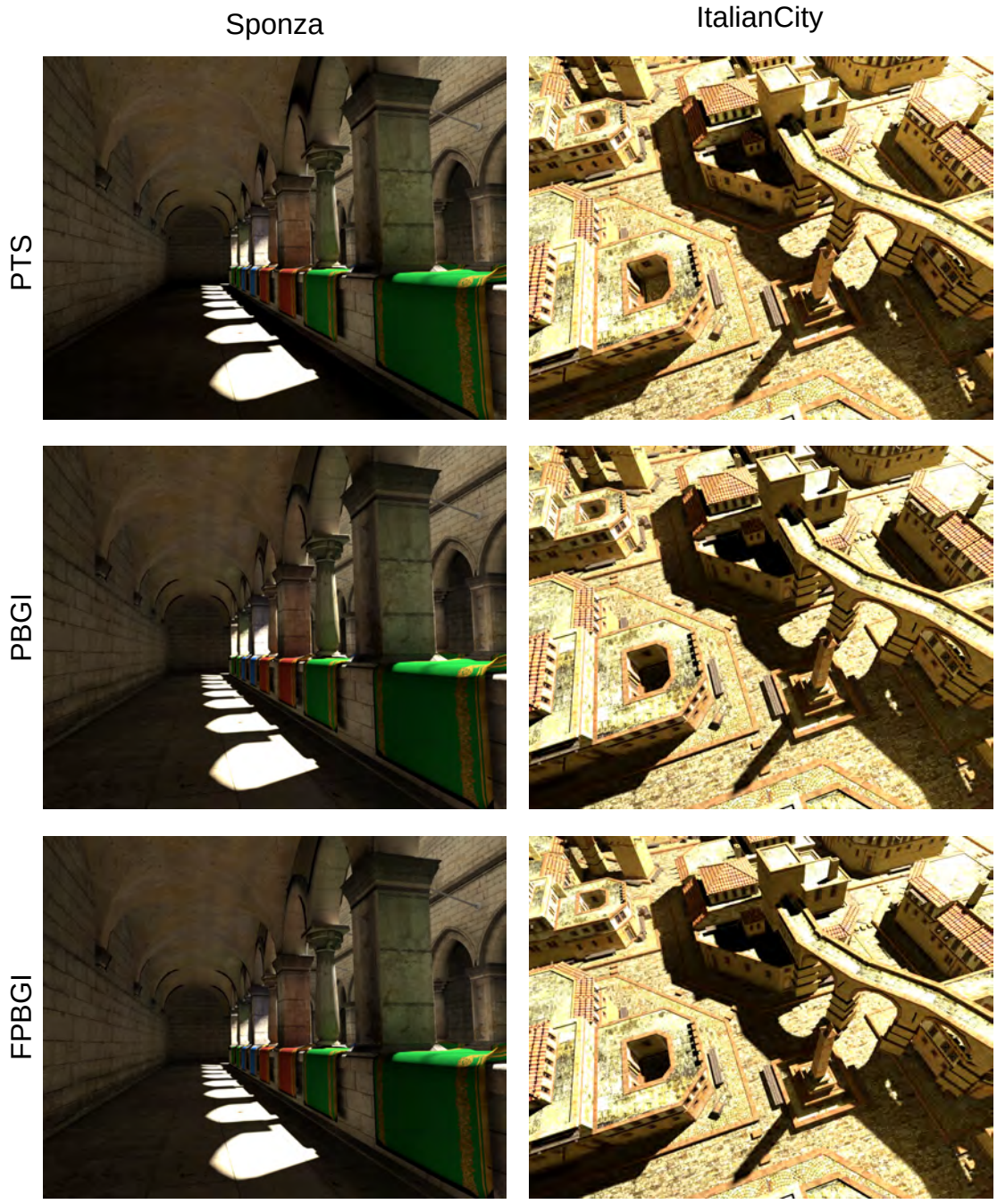


图 3.12 场景 Sponza 和场景 ItalianCity 最终渲染结果在 PTS，PBGI 和 FPBGI 间的视觉对比，图中的结果包括了直接光照和一次间接光照。

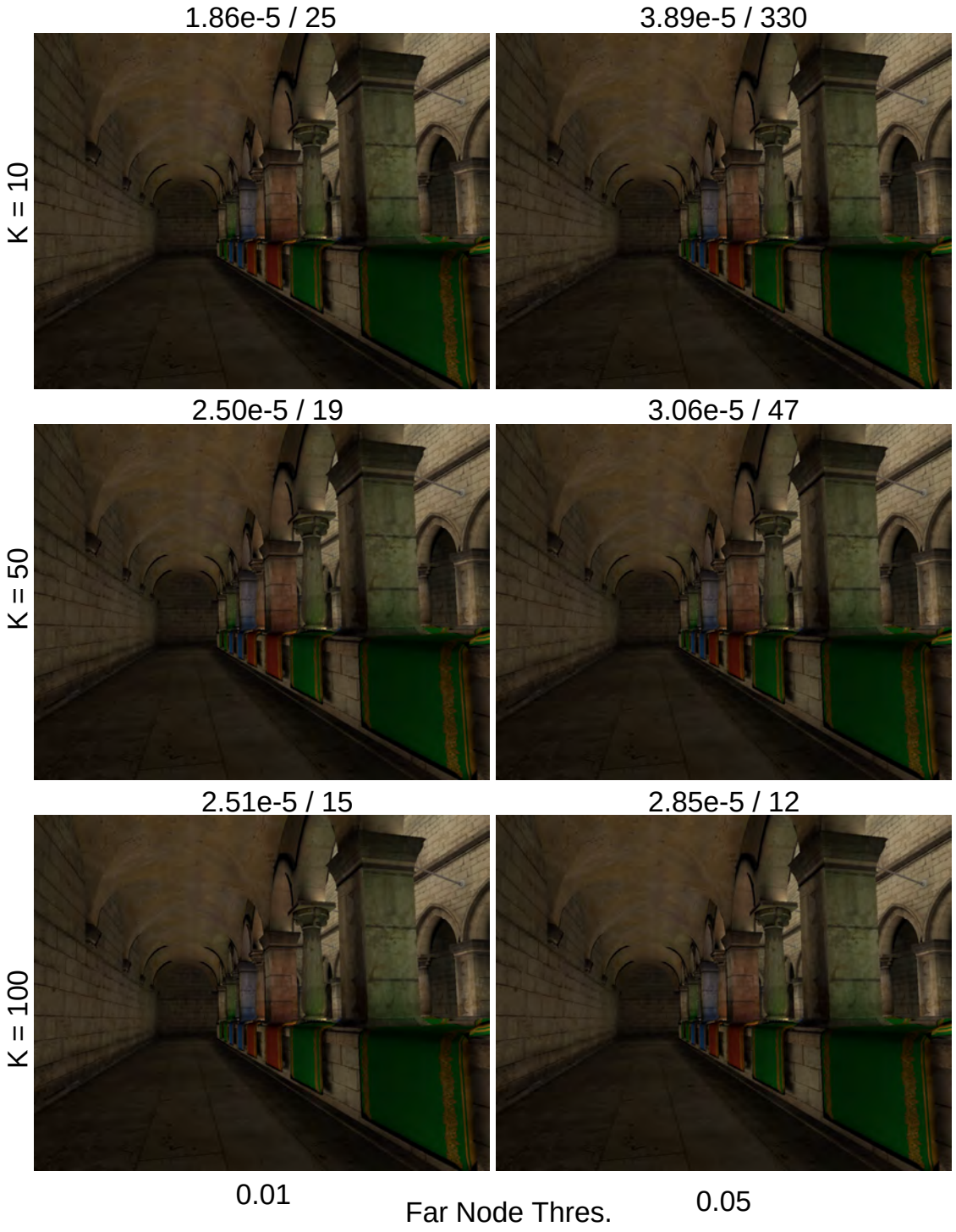


图 3.13 参数分析：在不同参数下的渲染结果，以及与 PBGI 算法进行对比的误差值<MSE>/<PDP>。

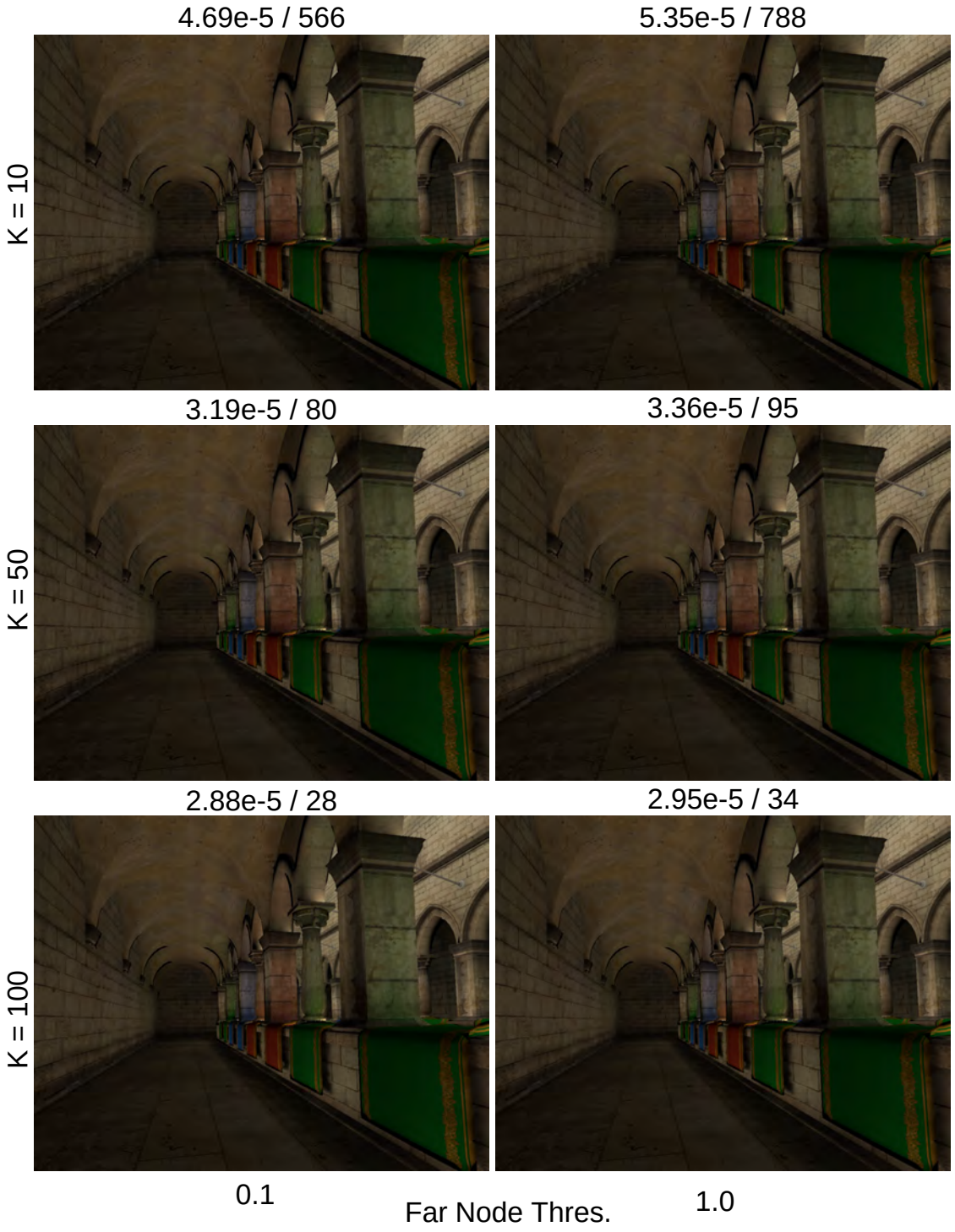


图 3.14 参数分析(续)：在不同参数下的渲染结果，以及与 PBGI 算法进行对比的误差值<MSE>/<PDP>。

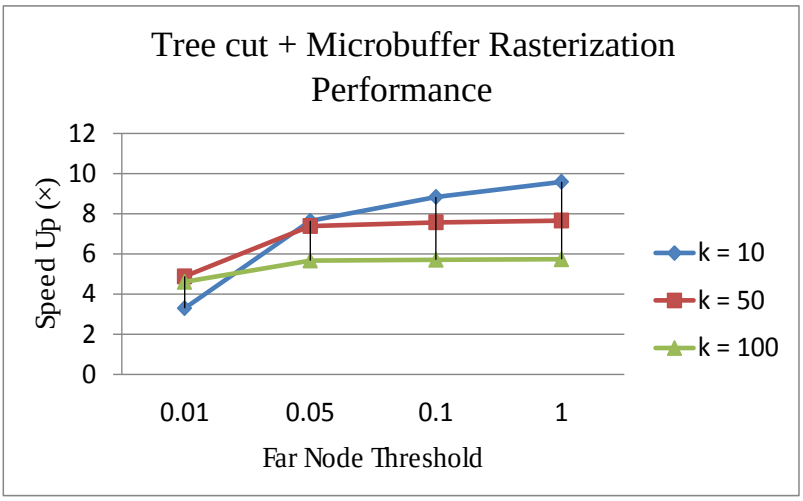


图 3.15 参数对加速比的影响分析折线图.

讨论. 关于 PBGI 的两种方法 [32, 36] 都提出最大化 PBGI 算法的并行性，从而将其映射到 GPU 上进行计算。 而我们的算法，虽然并没有使用 GPU 进行并行加速，但是保持了算法的天然并行性。 我们可以通过两种方法将 FPBGI 进行有效的 GPU 实现。 第一种，FPBGI 中聚类的数目足够多，从而可以充分利用 GPU 进行加速。 首先，在每个聚类间并行，每个线程计算聚类树切以及向聚类微缓冲区进行投影。 然后每个着色点进行并行，每个线程计算着色点树切以及着色点微缓冲区。 第二种，使用一种两层计算模型(blocks 和 threads)，使用属于一个 block 的线程来并行计算聚类树切和聚类微缓冲区。 同步之后，使用这些线程计算着色点的树切以及着色点微缓冲区， 此时可以使用 ManyLoDs [36] 中的方法，利用节点-着色点对来增加并行性。 Holländer 等人在[36] 中提出了基于时间连贯性的 *lazy* 模式，在连续的帧之间进行重用，我们的算法则是利用了空间连贯性，将这两种方法相结合，从而充分利用时间-空间连贯性，将是很有趣的工作。

我们的算法存在以下缺陷。 首先，聚类树切可能会过于细节(遍历到过低的节点)，这虽然不会影响到渲染结果的质量，但是仍然会影响到速度。 为了避免这个缺点，一种方案是着色点在遍历树的时候，当发生了过度遍历时，能够允许从下向上遍历。 其次，算法的两个主要参数是固定值，所以不能保证获得最优的效果。 最后，我们的算法其实是对于着色点层次的简化(只有一层即聚类)，而通过将着色点组织成层次化的结构，将会获得自适应的结果。

3.5 总结

在本章中，我们提出了基于分解的 PBGI 方案， 利用空间连贯性在聚类内的着色点重用树切和微缓冲区，从而提高间接光照的计算效率。 该算法保持了 PBGI 算法无噪声的优势，可以获得 2x 到 4x 的加速比，并且具有时间连贯性，易于在当前的 PGBI 框架中实现，并且只使用了少量可控的参数。

第 四 章　基于小波的点缓存全局光照

PBGI 可以用于模拟色溢、环境光阻挡、面积光、环境映射等效果，但是却无法模拟焦散、光泽反射表面之间的反射等非漫反射光线传递效果，这种缺点成为限制 PBGI 在电影动漫制作中使用的重要原因之一。

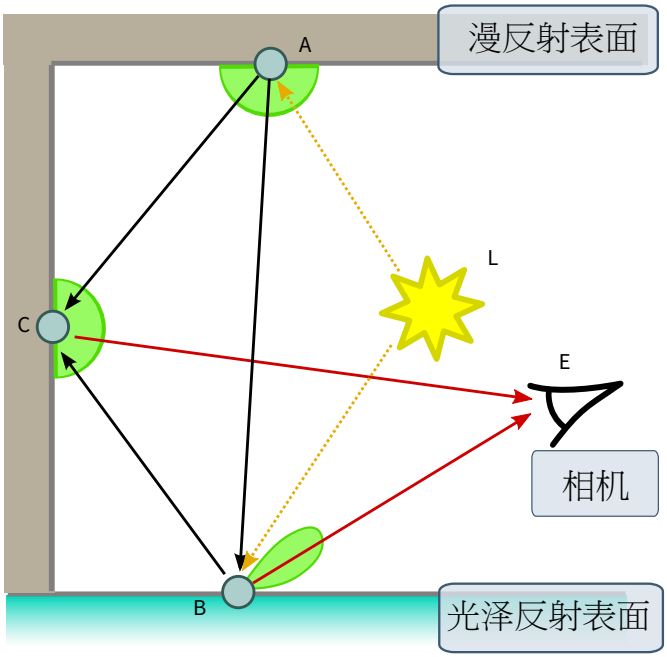


图 4.1　光源发射光线(黄色虚线)到场景中，与场景中的表面相交，部分能量被吸收，其他的部分被反射，反射的比例由材质的属性决定，反射的分布(绿色)根据表面 BRDF 的不同而不同，最终(初级射线为红色)到达相机中。以下使用字母组合表示光线的路径：LACE 代表的效果为色溢，LABE 代表的效果为光泽反射，LBCE 表示的效果为焦散。

图 4.1 表示了色溢，光泽反射和焦散的光线路径。PBGI 算法在点云缓存阶段，只存储了漫反射面的辐射亮度，符号表示为：LD(D|S)E。一次反射的焦散以及光泽表面之间反射表示为 LS(D|S)E。

已知 BRDF 表示光线如何在它所表示的材质上面发生的变化，在局部坐标系(z 轴与法向一致)中该函数可以参数化为 $\rho(\omega_i, \omega_o)$，其中，ω_i 表示光线入射方向，ω_o 表示光线出射方向。漫反射与视点无关，即任何出射方向(在法向半球中)的辐射亮度相同，因此，漫反射的 BRDF 可以重新参数化为 $\rho(\omega_i)$，维度从 4D

降低为 2D。 对于场景中的某点，如果光源固定，即 ω_i 是确定的值，那么，在 RGB 空间中，使用一个值即可表示出当前点向外的辐射亮度值。 非漫反射 BRDF 函数并不是视点无关的，因此如果要表示非漫反射点的出射辐射亮度，那么需要将 2D 函数表示出来，然后将其重构。

以上是出射辐射亮度在漫反射点和非漫反射点上表示的不同之处，因此，扩展 PBGI 计算焦散效果的核心是：缓存非漫反射表面的光线传递，即有效地表示二维函数。 “有效”包含了多层含义：如何既能表达高频函数又能表达低频函数；从内存使用角度来看，函数表示所使用的数据应该是紧凑的(compact)。

本章针对 PBGI 的以上缺点提出了基于小波的解决方案，从而使得 PBGI 有更广泛的应用。 此外，PBGI 在场景中出现强光或者高频 BRDF 时有明显走样， 通过增加微缓冲区的分辨率(减少立体角)可以解决该问题，但是这会导致速度大幅度降低， 从而失去了与光线跟踪算法相比的优势，本章中提出自适应的微缓冲区技术来解决走样问题。

在本章节中，我们首先讨论背景知识以及相关工作，然后着重提出基于小波的点缓存全局光照算法， 接着提出自适应的微缓冲区方案。

4.1 背景知识

在本节中，我们将基于 [47] 介绍关于哈尔小波 (Haar Wavelet) 的相关知识。哈尔小波是一种比较简单的小波基函数，由 Alfred Haar 在 1909 年提出。 本章中分别介绍一维哈尔小波和二维哈尔小波。

4.1.1 一维哈尔小波

令 V^n 表示所有的特定分段常量函数形成的子空间，这些分段常量函数定义在无重叠的 2^n 等分的子区间 $[0, 1)$ 上。 V^n 的一个标准正交基函数是 $\left\{\varphi^{(n,i)}\right\}_{i=0}^{2^n-1}$，其中 $\varphi^{l,i}(t) := \sqrt{2^l}\varphi(2^l t - j)$，并且

$$\varphi(t) := \begin{cases} 1, & t \in [0, 1), \\ 0, & \text{其他。} \end{cases}$$

基函数 $\varphi^{l,i}(t)$ 称为尺度函数，通过对父函数 $\varphi(t)$ 放缩和平移得到，其中，参数 l 用于控制放缩，i 用于控制平移，图 4.2 中表示了 V^2 空间下的基函数。

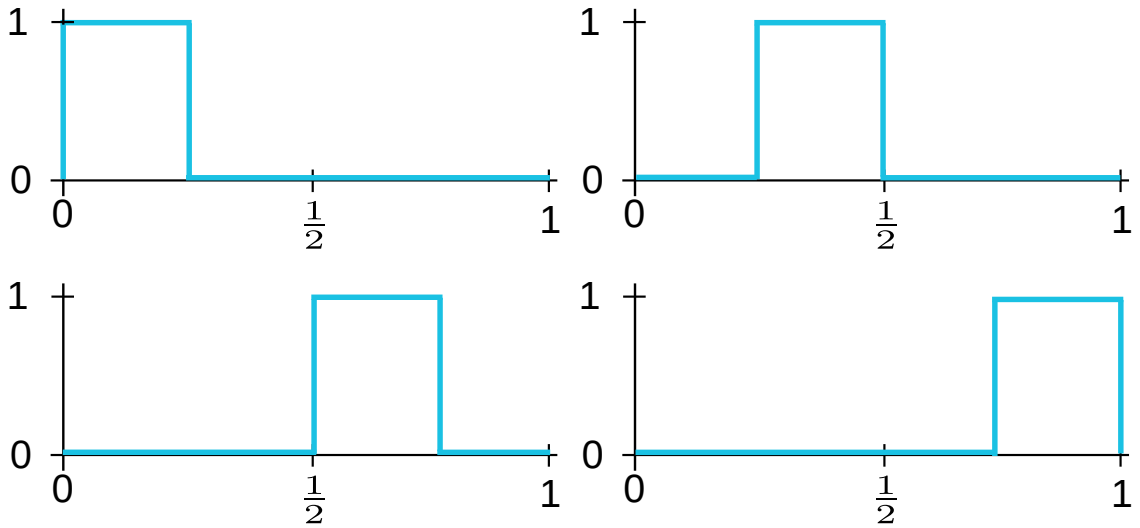


图 4.2 $\varphi^{(2,0)}$ (左上)，$\varphi^{(2,1)}$ (右上)，$\varphi^{(2,2)}$ (左下)和 $\varphi^{(2,3)}$ (右下).

函数 $\varphi(t)$ 是 V^0 的基函数，可以表示为 V^1 空间下基函数的线性组合：

$$\begin{aligned}\varphi(t) &= \frac{1}{\sqrt{2}}(\varphi^{(1,0)}(t) + \varphi^{(1,1)}(t)) \\ &= \varphi(2t) + \varphi(2t-1).\end{aligned} \tag{4.1}$$

从而有 $\varphi(t) \in V^1$。由于任意函数 $g \in V^0$ 都是 $\varphi(t)$ 的变形，因此 g 也是 V^1 中的一个元素，从而得到 $V^0 \subset V^1$，同理得到 $V^1 \subset V^2$，$V^2 \subset V^3$，以此类推。

令离散信号 f 是定义在 2^5 等分的 [0,1) 区间上的分段常量函数，那么 f 是 V^5 空间中的一个元素。令 f^n 是 f 在 V^n 上的投影，由于 V^l 上的基函数是标准正交的，因此该投影可以表示为

$$f^l(t) = \sum_{i=0}^{2^l} f_i \varphi^{(l,i)}(t),$$

其中，系数 f_i 表示 $\left\langle f, \varphi^{(i,j)} \right\rangle$，$\langle u, v \rangle := \int_0^1 u(x)v(x)dx$ 是标准内积。层次分解后的所有系数可以使用一个向量表示：

$$\left\{ \left\langle f, \varphi^{(0,0)} \right\rangle, \left\langle f, \varphi^{(1,0)} \right\rangle, \left\langle f, \varphi^{(1,1)} \right\rangle, ..., \left\langle f, \varphi^{(4,2^4-1)} \right\rangle \right\}. \tag{4.2}$$

但是以上系数并不是稀疏的。

令 W^n 表示 V^{n+1} 的子集，并且满足 $W^n \neq V^{n+1}$ 和 $W^n \cup V^n = V^{n+1}$。将定义在空间 W^n 上的线性独立的函数集定义为小波，对应于 V^n 空间上的小波基函数称为哈尔小波基函数。这些基函数是标准正交的，因此，W^n 是 V^n 在 V^{n+1} 中的正交补集。W^n 空间上的单位化小波基函数是 $\left\{\psi^{(n,i)}\right\}_{i=0}^{2^n-1}$，其中 $\psi^{(l,i)}(t) := \sqrt{2^l}\psi(2^l t + i)$，并且

$$\psi(t) := \begin{cases} 1, & t \in [0, 0.5), \\ -1, & t \in [0.5, 1), \\ 0, & \text{其他}。 \end{cases}$$

函数 φ 和 ψ 称为父小波和母小波，空间 V^n 上的基函数称为尺度函数。

已知 $V^n \cup V^n$ 与 V^{n+1} 共轭，因此 $V^n \cup W^0 \cup ... \cup W^n$ 与 V^{n+1} 共轭，因此空间 $V^i (i > 0)$ 上的基函数可以表示为小波和父小波的线性组合，即 f_i 可以使用 f_0 和 f 在 W^i 空间上的投影重构出来。由如下的系数向量 f' 即可重构 f，

$$f' = (\left\langle f, \varphi^{(0,0)} \right\rangle, \left\langle f, \psi^{(0,0)} \right\rangle, \left\langle f, \psi^{(1,0)} \right\rangle, ..., \left\langle f, \psi^{(4,2^4-1)} \right\rangle). \tag{4.3}$$

该公式与 (4.2) 类似，但是不同的是，在此公式中，很多系数值为 0 或者接近 0，即是稀疏的。

公式 (4.3) 可以一般化为

$$H(g) = (\left\langle g, \varphi^{(0,0)} \right\rangle, \left\langle g, \psi^{(0,0)} \right\rangle, \left\langle g, \psi^{(1,0)} \right\rangle, ..., \left\langle g, \psi^{(n,2^n-1)} \right\rangle). \tag{4.4}$$

4.1.2 二维哈尔小波

对于 2D 离散信号进行 2D 哈尔小波变换的最简单方法是标准分解，即先在行上进行 1D 哈尔小波变换，然后在列上进行。非标准的分解则是在行和列上交替进行变换。由于非标准 2D 小波变换可以更好适用于 2D 域，因此在应用中更多地被采用。

二维哈尔小波变换的小波基函数是

$$\begin{aligned} \psi_{01}^{(l,i,j)} &= 2^l \varphi(2^l x + i)\psi(2^l y + j), \\ \psi_{10}^{(l,i,j)} &= 2^l \psi(2^l x + i)\varphi(2^l y + j), \\ \psi_{11}^{(l,i,j)} &= 2^l \psi(2^l x + i)\psi(2^l y + j). \end{aligned} \tag{4.5}$$

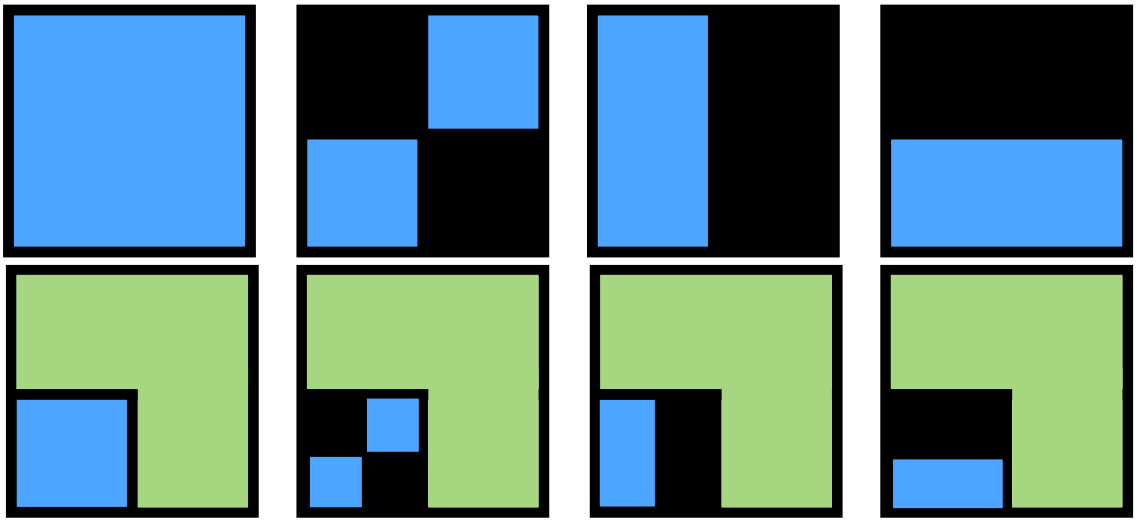


图 4.3 第一行从左到右依次为：$\varphi^{(0,0,0)}, \psi_{11}^{(0,0,0)}, \psi_{10}^{(0,0,0)}$ 和 $\psi_{01}^{(0,0,0)}$； 第二行从左到右依次为 $\varphi^{(1,0,0)}, \psi_{11}^{(1,0,0)}, \psi_{10}^{(1,0,0)}$ 和 $\psi_{01}^{(1,0,0)}$。 图中没有进行单位化，其中黑色区域表示函数的值为 -1，蓝色为 1，淡绿色为 0。

其中 l 表示小波的层，(i, j) 表示小波的位置。 尺度函数表示为

$$\varphi^{(l,i,j)} = 2^l \varphi(2^l x + i)\varphi(2^l y + j).$$

二维哈尔小波的例子在图 4.3 中表示。

4.1.3 哈尔小波的优点

哈尔小波基函数可以完美地将原信号重构出来。 傅里叶基函数与哈尔小波基函数的一个重要区别如下： 在前者是通过对 $cos\ nx$ 和 $sin\ nx$ 放缩得到， 而在后者中则是哈尔小波基函数通过对其父基函数平移和放缩得到的。 此外，哈尔小波基函数的支撑集（使得函数值为非零的自变量的集合）是局部的。 因此哈尔小波基函数在频率域和空间域上都具有局部性，而傅里叶基函数只在频率上具有局部性； 哈尔小波基函数具有支撑集局部性，使得它可以得到线性时间内的算法， 对于类似于 $sin\ nx$ 具有全局支撑集的基函数而言则有 $O(n^2)$ 的复杂性，即便是快速傅里叶变换也需要 $O(nlog(n))$ 的时间复杂性。 另外，哈尔小波基函数系数是稀疏的，即存在大量的系数值为 0 或者接近 0， 而傅里叶系数并不是稀疏的，当有大量数据需要存储时，哈尔小波系数是紧凑的，即需要的空间更少。

4.1.4 球谐函数

本节中，我们将基于 [48] 介绍球谐函数有关理论背景。 球谐函数 (SH) 广泛应用于球形函数的表示。 所谓的球形函数是指定义在球表面的函数，比如球坐标 (ϑ, φ)，其中 $0 \le \vartheta \le \pi, 0 \le \varphi \le 2\pi$。 SH 与傅里叶序列式类似，SH 用一组不同频率的正则基函数，而傅里叶序列则使用 $sin\ x$ 和 $cos\ x$ 作为基函数。 SH 中使用了连带勒让德多项式 $P_l^m(x)$，x 定义在区间 $[-1, 1]$ 上，l 是多项式的度，m 是多项式的阶， 其中 $l \ge 0$， $-l \le m \le l$。 l 可以理解为频率，当 $l = 0$ 时表示常数项。

球谐基函数 $Y_l^m(\varphi, \vartheta)$ 使用连带勒让德多项式 P_l^m 来表示。 虽然球谐函数定义为复数，但是我们只关注其中的实数部分。

$$Y_l^m(\varphi, \vartheta) = \begin{cases} \sqrt{2} N_{(l,m)} P_l^m(\cos\vartheta \sin(m\varphi)), & m > 0, \\ Y_l^0, & m = 0, \\ \sqrt{2} N_{(l,-m)} P_l^{-m}(\cos\vartheta \sin(-m\varphi)), & m < 0. \end{cases}$$

其中 $N_{(l,m)}$ 是正则项，从而使得 Y_l^m 为正则基。

任意的球形函数 $f(\varphi, \vartheta)$ 通过投影到以上的基函数上，即可得到在 SH 空间的表示。 所表示的函数是有限频宽的，频宽取决于最高度数 l，而高频部分被忽略。对于给定的度和对应的阶，系数 c_l^m 可以通过基函数 Y_l^m 和给定的函数 f 做内积来求得:

$$c_l^m = \int_0^{2\pi} \int_0^{\pi} f(\varphi, \vartheta) Y_l^m(\varphi, \vartheta) \sin\vartheta \, \mathrm{d}\vartheta \, \mathrm{d}\varphi.$$

函数 f 可以近似为系数与对应基函数乘积的和，表示为 $\hat{f}$，

$$\hat{f}(\varphi, \vartheta) = \sum_{l=0}^{L} \sum_{m=-l}^{l} c_l^m Y_l^m(\varphi, \vartheta). \tag{4.6}$$

4.2 相关工作

4.2.1 基于小波的渲染

小波在渲染领域得到广泛的应用，比如小波辐照度算法 [49, 50]。 在 [50] 中，Christensen 等人使用小波基函数表示场景中表面的辐射亮度分布，并且根据相关条件判断是否需要更新， 比如是否需要对面片进行细分，或者是否需要在方

向上进行细分 (对于非漫反射的表面)， 从而体现出小波基函数在多分辨率分析 (multiresolution analysis) 中的优势。

另外小波大量应用在预计算辐射亮度传输 (precompute radiance transfer，简称为 PRT) 算法中。 该类算法假设场景中几何物体不变，对部分的光线传递进行预计算，从而使得渲染阶段可以在可交互效率或者实时下进行重照明 (relighting)，或者高效计算色溢、焦散等复杂的效果。 Sloan 在 [51] 中提出第一个 PRT 算法，在预计算阶段，使用球谐函数分别表示光照传输 (BRDF 和可见性) 以及环境映射；在渲染阶段，将 SH 系数相乘得到最终的辐射亮度。

之后，小波用于光照函数和 BRDF 的表示。 Ng 等人在 [52] 中提出，预处理阶段使用小波系数表示环境映射，并且对每个着色点的 BRDF (在视点可变的情况下只允许漫反射材质，从而使得 BRDF 参数化为 2D) 使用小波系数表示； 在渲染阶段，将两者相乘得到着色点的辐射亮度。 在存储系数时，对小波系数进行非线性近似，即只存储一定比例的值较大的系数，从而降低内存使用。 [53] 与 [52] 相比， 分别将可见性，光照，以及 BRDF 进行小波分析，并且提出将三组小波系数快速相乘的方法。 与之前的方法相比，效率更高，并且在视点可变的情况下允许非漫反射的材质。 以上两种方法在对 BRDF 进行小波分析时， 都首先通过立方体贴图 (cube map) 中的每个方向进行采样，然后对每个面分别进行哈尔变换。 本章也采用类似的方法。 Sun 等人在 [54] 中对以上的方法进行改进， 将三组小波系数相乘扩展到大于三的情况，从而使其支持动态场景下的可见性判断。 Kontkanen 等人 [55] 提出基于辐照度方法的 PRT 算法。 在预处理阶段使用小波系数表示 4D (位置为 2D ，法向为 2D) 的全局光照传递算子 (global transport operator)，该算子作用于直接光照，从而得到光照在场景中的传输 (多次反弹)。 在渲染阶段，首先将直接光照在相同的基函数上进行投影，然后将全局光照传递算子系数作用于直接光照系数从而得到表示入射辐射亮度的系数， 继而与该点表示 BRDF 的小波系数相乘得到出射辐射亮度。 该方法最大的贡献在于支持任意局部光照，而在此之前的三个基于小波的 PRT 算法只支持环境映射作为光源。 [56] 中也将光照函数和传递函数在小波空间进行表示，与 [55] 不同的是，小波基函数并不是定义在面片 (patch) 上，而是定义在对面片进行采样后产生的点云上，这种 mesh-less 的思路相对更加灵活，不需要考虑细分等。 本章中提出的方法，也将小波定义于点云上，

但是与该方法不同的是我们将函数在方向变量上进行参数化，从而支持非漫反射光线传递。

此外，小波分析也广泛应用于 BRDF 建模中，比如 [57] 和 [58] 中。前者使用不同的小波基函数 (哈尔基函数、线性样条基函数和 Daubechies 基函数) 表示捕捉的 BRDF 数据 (4D，出射方向和入射方向分别为 2D)。后者提出基于小波的通用模型，该模型为 5D (其中波长为 1D)，该模型将谱部分同几何部分分开，从而使得表示更加紧凑。

4.2.2 其他的基函数

除了使用小波来表示渲染相关函数外，还有其他的基函数，比如 (半) 球谐函数((Hemi)Spherical harmonics (SH / HSH))。在 [59] 中，SH 系数用于存储环境映射的辐射照度。在 [51] 中，SH 系数用于表示传递函数和光照函数，该方法也是最早提出的 PRT 方法。目前广泛应用于全局光照算法中的辐射亮度缓存算法 [42] 也采用了 SH (HSH) 系数缓存采样点入射辐射亮度和 BRDF，在着色点处对入射辐射亮度系数进行插值，再与 BRDF 相乘就得到着色点出射辐射亮度。在该方法中，HSH 只能表示频率较低的 BRDF，对于频率高的情况则采用其他的方法，原因在于频率较高的情况需要大量系数才能使得重构出来的函数与原函数误差较小，这将导致内存问题。在 [24] 中，SH 系数用于表示点云层次结构中每个节点的出射辐射亮度，该方法只支持非漫反射光线传递，每个节点所需要表示的辐射亮度分布相对平滑，因此采用较少的系数就可以将原函数表示出来，文中使用了 9×3 (RGB) 个系数用来表示一个节点的出射辐射亮度。SH 定义在球空间，相对哈尔小波在表示 BRDF 等球形函数时更具有优势，此外 SH 支持旋转，这对于空间之间的变换提供了很多方便，而哈尔小波不支持。但是，SH 有以下缺点：首先，能够高效地表达低频函数，但是对于高频函数，却需要大量的系数作为代价；由于不支持稀疏表示，因此会导致严重的内存问题；使用 SH 表示高频函数时还存在 “ringing” 走样，该问题虽然可以通过 sinc 等窗函数滤波来减少，但是很难消除；最后，SH 是傅里叶函数在球上的表示，也具有全局性支撑集的特点，因此在进行重构时与小波相比需要更多的处理时间。

球形高斯 (SGs) 也广泛的应用在渲染相关的函数表示中，比如光线传递函数，BRDF 等。 SGs 是球形径向基函数 (Spherical Radial Basis Function，简称为 SRBF) 的一种，它的模型表示为

$$G(v; p; \lambda, \mu) = \mu e^{\lambda(v \cdot p - 1)}.$$

其中 $p \in \mathbb{S}^2$ 是 lobe 轴，$\lambda \in (0, +\infty)$ 表示 lobe 的尖锐程度 (sharpness)，$\mu \in \mathbb{R}$ 表示 lobe 的振幅 (amplitude) (在 RGB 空间中，$\mu \in \mathbb{R}^3$), 方向 $v \in \mathbb{S}^2$ 表示函数的球形参数。将多个 SGs 相加之后，即可得到球形高斯混合模型，用来表示几个 lobe。 [60] 中，使用高斯混合模型表示光线传递函数(包括 BRDF 和可见性)， 由于球形高斯基函数无法表示细节较多的可见性，因此该方法只支持软阴影。 在 [61] 中，只使用高斯函数来表示 BRDF，而可见性则使用 SH 来表示。 Wang 等人在 [62] 中使用高斯函数来表示 BRDF 和 cosine 项，以及点光源的光照，使用其他的方法来表示可见性以及环境映射等。 Xu 等人在 [63] 中提出了各向异性 SG (AnisotropicSG) 来表示各向异性的 BRDF 函数。 [64] 中使用 SRSFs 来表示环境光照，其中的 BRDF 采用了分解表示的方法。 SGs 定义在球形上，并且支持旋转，在表示 BRDF 时表现出很大优势，需要的存储较小， 但是该函数不适合表示细节较多的函数，比如环境光照以及表示着色点的入射辐射亮度等。 另外，球形高斯混合模型并不是正交的，在计算两个模型的内积时，需要考虑所有的 SG 对，即复杂性为 $O(n^2)$，其中的 n 为 SG 的个数。

4.3 算法概述

在本章中，我们提出基于小波的 PBGI (*Wavelet PBGI* 或者 *WPBGI*) 来模拟非漫射光线的传输，比如焦散。 首先提出辐射亮度小波表示模型，接着提出小波系数层次化编码技术来解决存储问题， 最后提出自适应微缓冲区，在包含经典几何因素 (距离、立体角) 的基础上，引入了光照和 BRDF 的光泽度 (glossiness) 等因素，算法的流程如 4.4 所示。

我们的算法能够渲染焦散等非漫反射传递特效，支持的 BRDF 频率从漫反射到到非常接近于完全镜面的反射， 并且相比双向路径跟踪 (BPT) 和渐进式光子映射 (PPM) 具有数倍的提速。

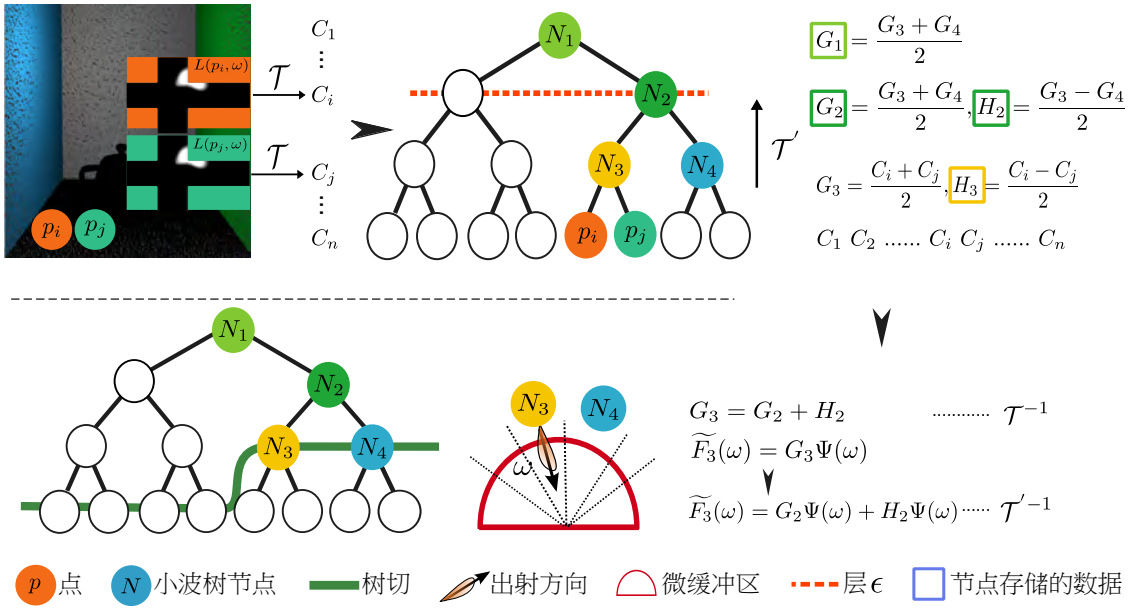


图 4.4 算法流程。 首先，对场景采样得到点云，将点云组织到点云层次结构，并且采用后序遍历来计算每个节点/点的小波系数。对于叶节点/点，将其直接光照按照立方体贴图采样，然后对立方体贴图的每个面进行哈尔小波变换 ($\mathcal{T}$) 从而得到系数向量 C_i；计算中间节点的辐射亮度分布，对子节点小波系数进行小波分析 ($\mathcal{T}'$，或者说求和平均和求差平均)，从而得到两种系数类型，节点近似系数 G 和细节系数 H。树中小于(靠近根节点)层 ϵ (红色虚线)的节点只存储近似系数，层数大于(靠近叶节点) ϵ 的节点只存储细节系数，该层上的节点存储两者；计算着色点的间接光照时，遍历点云树，将节点投影到微缓冲区中，首先计算出节点近似系数(大于 ϵ 层的节点)，然后，重构出节点的辐射亮度。

4.4 小波辐射亮度模型

在 PBGI 算法中，点云中的每个点缓存了点的位置、法向、半径以及漫反射的颜色，由于未存储非漫反射表面的辐射亮度信息导致 PBGI 算法无法模拟焦散效果。在本节中，我们提出小波辐射亮度模型用于缓存非漫反射光线的传递。

4.4.1 小波辐射亮度模型

将点云中某一点 p 的辐射亮度分布参数化为 $L(x,\omega_o)$，其中 x 表示位置，ω 表示出射方向，如图 4.5 所示。在 PBGI 中，每个点都需要计算分布，给定点 p_j，其辐射亮度分布可以参数化为 $L_j(\omega)$。

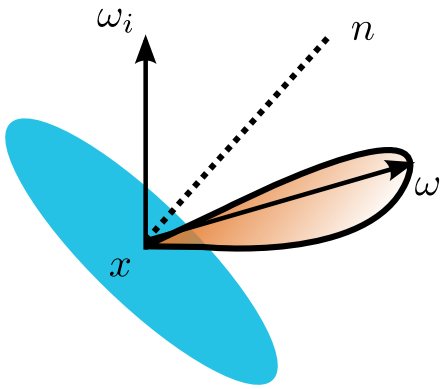


图 4.5 位置为 x、法向为 n 的某点在全局坐标系下的参数表示。 ω_i 表示入射光线方向，ω 表示出射光线方向， 渐变红色表示了该非漫反射点出射辐射亮度的分布情况。

该函数在基函数 B 上投影从而表示为一系列的展开式

$$L_j(\omega) = \sum_{i=1}^{\infty} c_i b_i(\omega),$$

其中，$b_i(\omega)$ 表示基函数，c_i 表示系数。

哈尔小波在空间和频率域上都具有局部性特点，并且支持非线性近似， 因此，我们使用哈尔小波函数作为基函数。

4.4.2 小波系数求解

在每点 p_j 定义全局坐标系下的立方体贴图 (比如分辨率为 $6 \times 32 \times 32$)，然后将函数 $L_j(\omega)$ (直接光照) 在此立方体贴图每个方向上进行采样，从而得到对应于立方体贴图六个面的六张图片。 对每一张图进行二维哈尔变换 (非标准变换)，并将得到的系数用树结构存储，类似于 Sun 等人在 [54] 中提出的方法，将在小节 4.4.3 中详细介绍。 采用树结构来存储小波系数，而没有使用更加紧凑的方法 (比如零树 [65]) 原因在于，重构时需要进行大量的点重构，即给定某个方向，使用小波系数计算出在这个方向上的辐射亮度， 而这些高效的结构以及算法的前提是把整个图片都重构出来，不适用于我们的情况。

每个点的辐射亮度分布用这些系数来表示，点 p_j 的系数集合用 $C(j)$ 来表示。

4.4.3 小波系数的存储模型

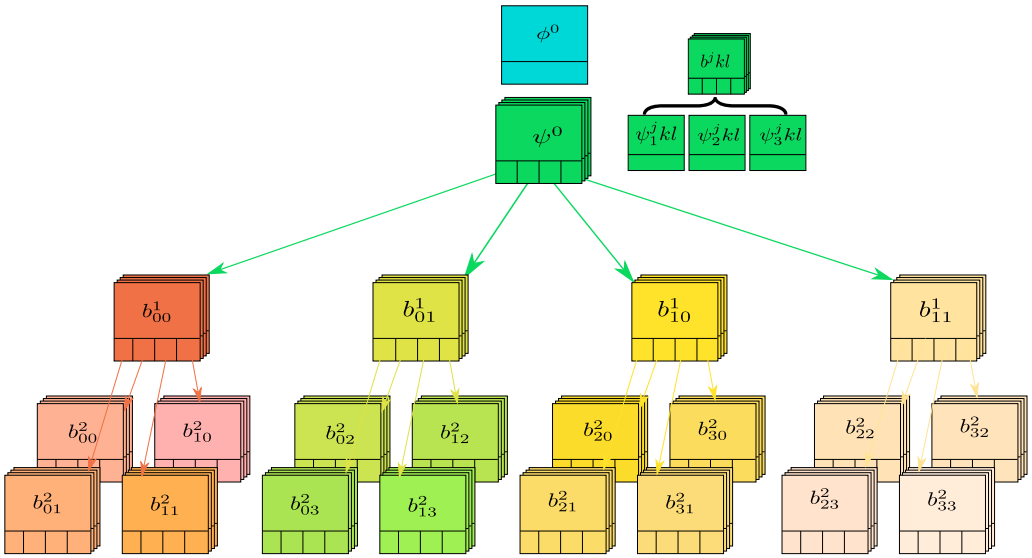


图 4.6 2D 哈尔小波系数的存储模型。 哈尔小波树存储母尺度基函数的系数(蓝色)以及最高层的小波基函数系数节点(绿色)。 小波基函数系数节点，包含了三个小波基函数的系数，以及四个子节点的指针。

在存储小波系数时，采用类似于 [57] 和 [54] 中的树结构模型， 如图 4.6 所示。在实现中，立方体贴图的每个面使用树结构存储，即包含了尺度函数系数以及表示小波函数系数的根节点。 每个节点存储了三个小波基函数的系数以及四个子节点的指针。

为了避免将点云中所有点的小波系数存储在内存中，在点云层次结构构建阶段采用 “on the fly” 策略， 即后续遍历 (post order) 方法，将在小节 4.5 中详细介绍。

4.5 小波树层次化编码

小节 4.4 中介绍了每个点的小波模型，在本节中，我们将提出小波树模型，以及小波系数的层次化编码技术。

4.5.1 小波树

小波树是 BSH 的变形，也是二叉树，自底向上构建，每个叶节点中包含了一个点，中间节点包含了它所近似子树的信息，比如位置、法向、半径以及辐射亮

度分布。 在 PBGI 算法中，每个中间节点的辐射亮度分布用球谐函数来表示， 基于小节 4.2 中的分析，本章中将使用小波代替球谐函数。

小波树中，节点 n_j 的辐射亮度分布参数化为 $Ln_j(\omega)$，定义如下：

$$Ln_j(\omega) = \begin{cases} L_j(\omega), & \text{节点 } n_j \text{ 是叶节点,} \\ \dfrac{Ln_{j^-}(\omega) + Ln_{j^+}(\omega)}{2}, & \text{其他。} \end{cases}$$

其中 $Ln_{j^-}(\omega)$ 和 $Ln_{j^+}(\omega)$ 分别表示节点 n_j 的左右子节点的辐射亮度分布函数。

4.5.2 小波系数编码

在构建小波树时，采用自底向上的策略，因此在计算某节点的小波系数之前，其子节点的系数已经被计算出来， 叶节点的小波系数来自于其包含点的小波系数。每个点的小波系数，采用全局坐标系，因此可以在不同点/节点之间的系数向量之间直接进行线性运算，不需要旋转。我们用 $G(j)$ 表示中间节点 n_j 的系数向量，其计算公式如下：

$$G(j) = \frac{G(j^-) + G(j^+)}{2}.$$

其中 $G(j^-)$ 和 $G(j^+)$ 分别表示左右子节点的小波系数向量。

考虑到存储的充分性，我们将节点的小波系数按照层次结构进行编码。 对节点的小波系数定义两个编码操作：求和平均，求差平均，如下:

$$G(j) = \frac{G(j^-) + G(j^+)}{2}, \quad H(j) = \frac{G(j^-) - G(j^+)}{2}.$$

其中 G_j 称作节点近似系数，H_j 称作节点细节系数。

对于小波树进行节点小波系数编码的过程，就是对点云系数进行小波分析 (1D 哈尔变换) 的过程。

我们提出以下策略存储节点小波系数。 首先，我们对节点定义"层"，根节点的层为 0，向下依次递增。 将树从 ϵ 层进行划分，大于 ϵ 层的节点，只存储节点细节系数； 小于 ϵ 层的节点，存储节点近似系数； 位于层 ϵ 上的节点，存储这两种系数，ϵ 是用户预设值。

层数较大的节点和邻居节点 (兄弟节点) 更加相似， 因此节点细节系数相比于节点近似系数中的数值可能更小，更有利于非线性近似计算。 随着层数的减少，邻居节点之间的连贯性降低，该特点将不存在。

为了减少内存使用，对小波系数进行非线性近似计算，即删除掉绝对值小于某个阈值的系数 原因在于信号大部分能量存储在一小部分具有较大绝对值的系数中。

4.5.3 节点辐射亮度重构

计算节点 n_j 在方向 ω_o 上的辐射亮度时，使用节点近似系数与小波基函数卷积得到即可，

$$Ln_j(\omega_o) = G(j) \otimes B \tag{4.7}$$

但是层数大于 ϵ 层的节点中并没有存储节点近似系数，因此需要先将其计算出来:

$$G(j) = G(m) + \sum_{k=0}^{l} s(k) \times H(k).$$

其中 m 表示从根节点到节点 n_j 路径上位于层 ϵ 上的节点索引， s 表示节点的符号：左节点为 1，右节点作为 -1，k 表示从根节点到节点 n_j 路径上节点的索引， l 表示节点 n_j 的层。

4.5.4 实现细节

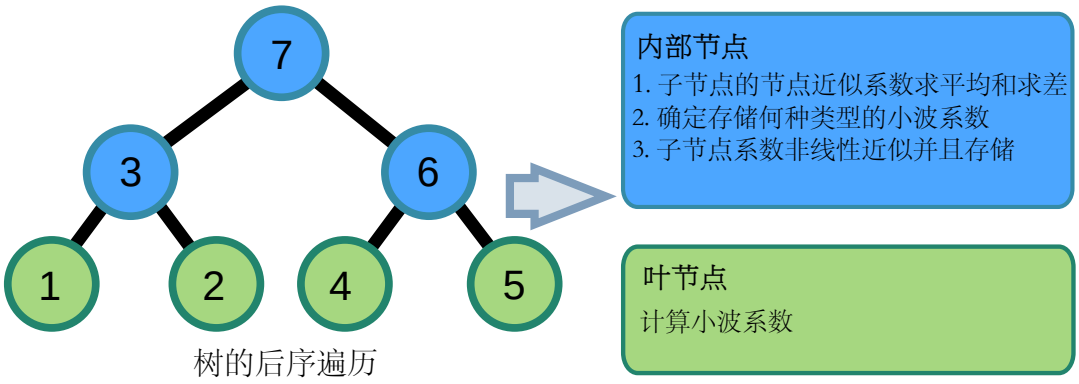


图 4.7 为了避免整个点云的小波系数都存在内存中，我们采用后序遍历的方式，on-the-fly 模式计算点云的小波系数： 计算子节点的小波系数并且进行和平均、差平均，对子节点决定存储何种小波系数，进行非线性近似并且存储。

在构建小波树时，采用后序构建的方法，如图 4.7 所示。对每个叶节点，按照 4.4 中的模型来计算点的小波系数；而对中间节点，计算它的两个子节点的小波系数平均和、平均差，即节点近似系数和节点细节系数， 然后根据节点层数决定

Algorithm 1: Wavelet Tree Construction

```
Data: BSH
Result: WaveletTree
begin
    nodes ← post_order(tree)
    foreach n ∈ nodes do
        if n is leaf then
            C_n ← compute_wavelet_coefficients(n)
        else
            G_n ← average_G_n.children
            H_n ← difference_G_n.children
            foreach c ∈ n.children do
                if c.l < ϵ then
                    c.data ← non_linear_approximation(G_c)
                else if c.l == ϵ then
                    c.data ← non_linear_approximation(G_c)
                    c.data ← non_linear_approximation(H_c)
                else
                    c.data ← non_linear_approximation(H_c)
```

存储其子节点的何种类型的系数，将子节点的小波系数非线性近似后存储。这样可以避免多于 $q+1$ 组未压缩的节点系数存在内存中，其中 q 表示树的度 (比如，BSH 的 q 为 2，而八叉树的 q 为 8)。为了防止自底向上模式中累计近似造成误差的放大，对于每个节点使用其子节点的未近似系数后，再对其子节点系数进行压缩。算法描述如算法 1 所示。

4.6 重要性驱动微缓冲区

4.6.1 理论基础

辐射亮度计算公式如下：

$$L(x,\omega_o)=\int_{\Omega_{2\pi}} L_i(x,\omega_i)\rho(x,\omega_i,\omega_o)V(x,\omega_i)(\omega_i\cdot n)d\omega_i. \tag{4.8}$$

其中 L 表示辐射亮度，是关于位置 x、出射方向 ω_o 的函数， ω_i 表示入射方向，ρ 表示 BRDF，V 表示可见性项，n 表示面的法向， L_i 是在 ω_i 方向上的入射辐射亮度。

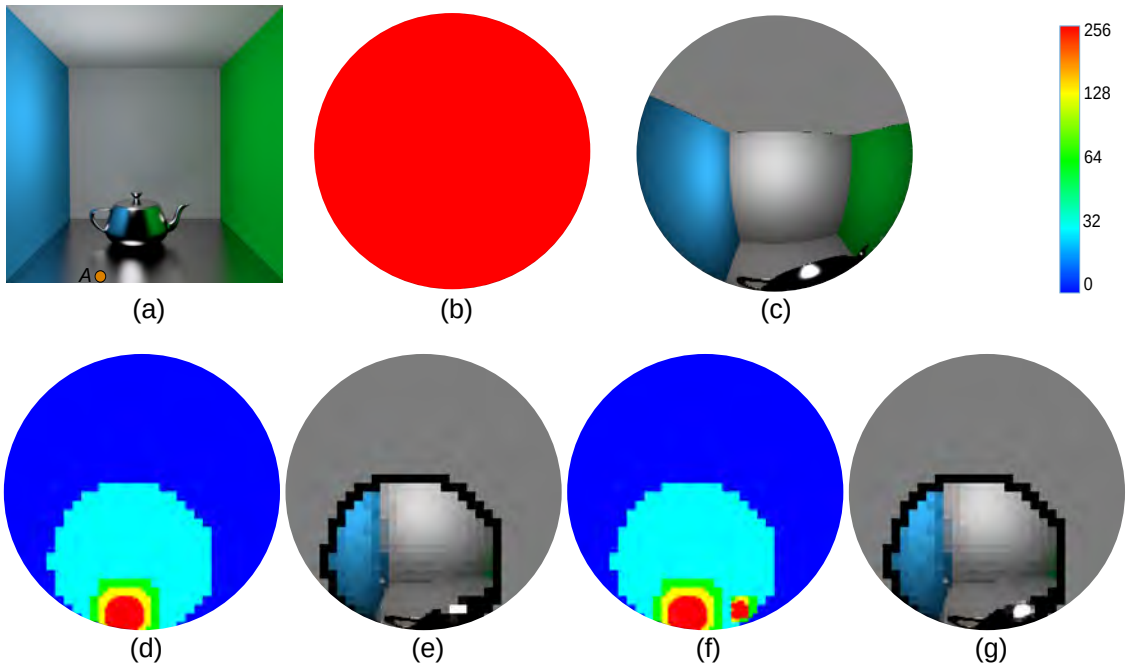


图 4.8 重要性驱动微缓冲区图示。(a) 在场景中有一个光泽反射的着色点 A， 在 (b) 和 (c) 中，构建规则 (uniform) 微缓冲区，其中 (b) 可视化了规则微缓冲区的分辨率，(c) 中可视化节点投影后的颜色信息； (d) 和 (e) 中分别表示在使用 BRDF 驱动后 AMB 的分辨率，向 BRDF 重要性驱动的 AMB 进行投影后的结果； (f) 和 (g) 表示在 BRDF 驱动基础上再进行入射光线辐射亮度重要性驱动后 AMB 的分辨率，以及点云层次结构中的节点向该微缓冲区投影后的结果。

在 PBGI 中，该方程改写为

$$\widetilde{L}(x,\omega_o) = \sum_{k=0}^{M} L_i(x,\omega_k)\rho(x,\omega_k,\omega_o)V(x,\omega_k)(\omega_k \cdot n). \tag{4.9}$$

其中 $\widetilde{L}$ h:ΞfiM /′:∗pfiω_k 是微缓冲区像素 k 的出射方向。 L_i 是投影到微缓冲区该像素中节点的辐射亮度，根据公式 4.7 求解。

根据公式 (5)，我们提出了重要性函数驱动的微缓冲区构建方法，该重要性函数考虑了 BRDF 和入射光照，该微缓冲区称为自适应微缓冲区 (AMBs)。 自适应微缓冲区的出发点在于，微缓冲区不一定是规则的，只需要在重要方向使用高分辨率。 重要性函数定义如下：

$$\mathcal{F}(k) = \mathcal{F}_l(k)\mathcal{F}_\rho(k).$$

其中 k 表示微缓冲区像素的索引，$\mathcal{F}_l$ 是基于入射光照的重要性函数，$\mathcal{F}\rho$ 是基于 BRDF 的重要性函数。计算$\mathcal{F}\rho$ 采用类似于 [32] 中提出的方法，不同之处在于我们没有进行变形 (warp)，而是对像素进行细分。 $\mathcal{F}_l$ 的定义如下：

$$\mathcal{F}_l(k) = \begin{cases} 1, & \dfrac{L_i(k)}{d^{\Delta_k}} > \gamma, \\ 0, & \text{其他。} \end{cases}$$

其中 d 表示细分时的维度，Δ_k 表示在光照重要性驱动下像素 k 在层数上的增量，γ 表示阈值。

4.6.2 实现细节

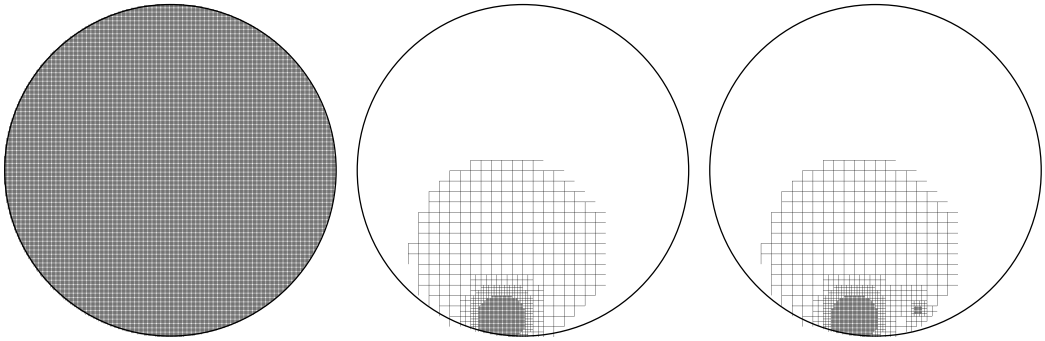


图 4.9 微缓冲区采用数据结构的可视化，(a) 为规则微缓冲区，(b) 是基于 BRDF 重要性驱动的结果，(c) 是在使用了基于光照的重要性驱动更新后的结果。

在实现中，AMBs 组织为树，该树中每个节点有固定像素的微缓冲区 (比如 2×2)，即四叉树。 微缓冲区初始化为规则的缓冲区，比如 (16×16)，然后根据 $\mathcal{F}_\rho$ 将其更新。 接下来，根据微缓冲区的每个像素所产生的立体角来遍历小波树， 同时根据 $\mathcal{F}_l$ 来判断当前的像素是否需要进一步细分。 在遍历的过程中，如果当前点云树节点的立体角满足当前像素的立体角， 那么该节点存储在微缓冲区像素的链表中，而不仅仅存储距离着色点最近的点云树节点。 如果某个像素需要进一步被细分，那么所有存储在当前像素中的点云树节点要重新进行遍历和投影。 在遍历和投影阶段之后，对距离很近的点云树的叶节点使用光线投射 (ray casting)。 最后，着色点微缓冲区中的入射光照与每个像素的 BRDF 还有 cosine 值卷积后得到最终的间接光照， 如算法 2 所示。

Algorithm 2: Rendering Process

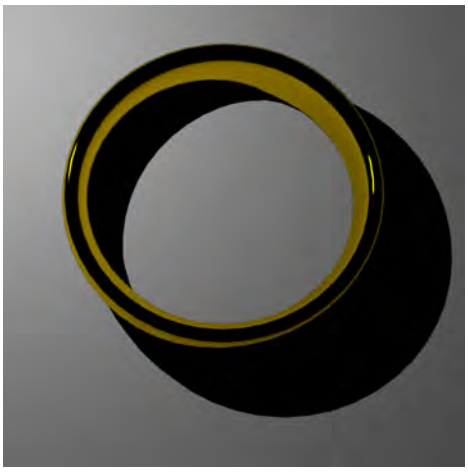
```
Data: WaveletTree, Receiver
Result: Outgoing Radiance
begin
    mb ← init_microbuffer_BRDF(receiver)
    node_queue ← push(tree.root)
    while node_queue.not_empty() do
        node ← node_queue.pop()
        if node is leaf then
            splat_leaf(mb, node)
        else
            solidangle ← evaluate_solidangle(node, receiver)
            solidangle_thresh, pixel ← splat(mb, node)
            if solidangle <= solidangle_thresh then
                F_l ← evaluate_light_importance(node, receiver)
                if !F_l then
                    pixel ← splat(mb, node)
                    pixel.node_list ← push(node)
                else
                    node_queue ← push(pixel.node_list)
                    node_queue ← push(node)
            else
                node_queue ← push(node.children)
    outging_radiance ← convolve(mb, receiver.brdf)
```

4.7 实验结果

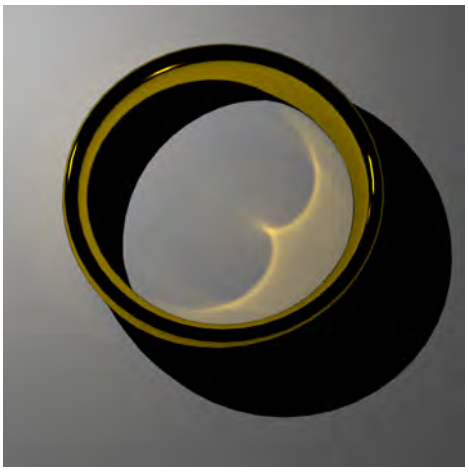
4.7.1 准确性验证及对比

我们的技术在 Mitsuba Renderer [2] 上进行实现，与 PBGI 算法、渐进式光子映射算法 (PPM) 以及双向路径跟踪 (BPT) 算法进行对照，在所有的对照中，我们使用均方差 (MSE)。实验的硬件环境是 2.67 GHz、Intel i7 (8核) 处理器，9 GB 内存。所有的渲染结果计算了一次反弹的间接光照，分辨率为 1024 × 768 (Ring 场景和 Cornel Box 场景除外，使用了 1024 × 1024 分辨率)。对图片进行 32 × 32 分块后并行渲染，自适应采样用来反走样。

PBGI vs WPBGI. 在图 8 中，将 WPBGI 算法和 PBGI 算法对不同场景的渲染效果进行对比，我们发现 WPBGI 算法能够模拟非漫反射光线传输效果，比如焦散，



PBGI, 2.39h, 2.32GB



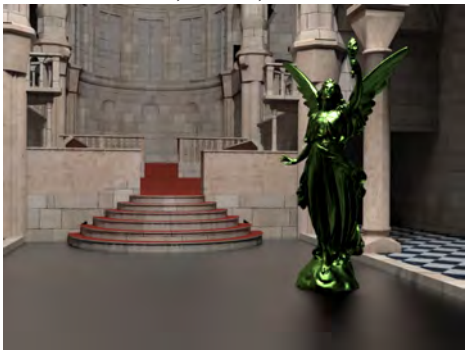
WPBGI, 1.25h, 5.98GB

PBGI, 13.08h, 3.57GB

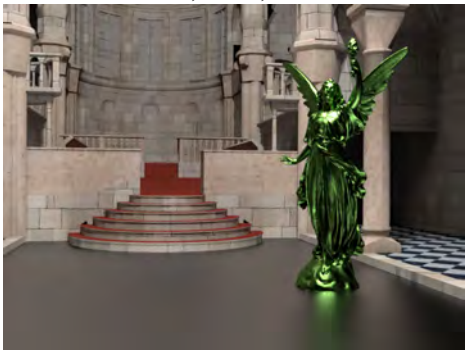
WPBGI, 1.63h, 6.62GB

PBGI, 10.50h, 1.93GB

WPBGI, 1.04h, 5.68GB



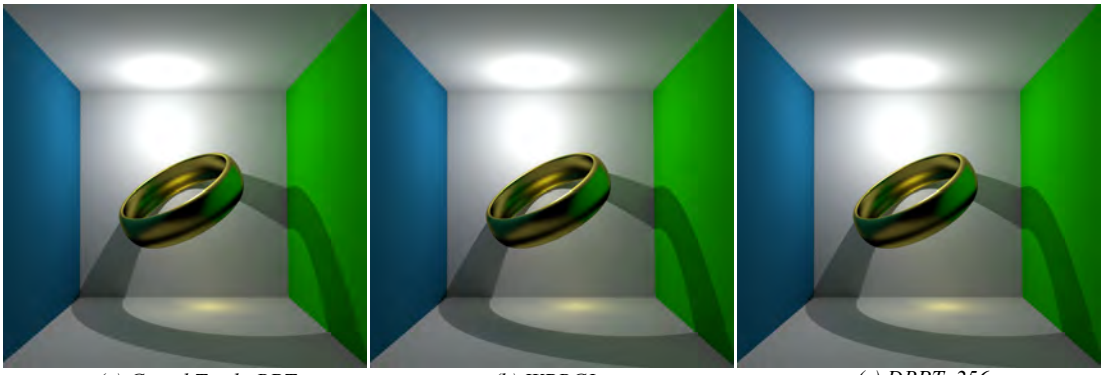
PBGI, 9.16h, 4.41GB



WPBGI, 1.94h, 5.85GB

图 4.10 PBGI 与 WPBGI 对比

而 PBGI 算法则无法计算。 另外，由于在 WPBGI 中使用了重要性驱动微缓冲区，在支持非漫反射光线传输的同时，所需要的渲染时间相比 PBGI 更少。



(a) Groud Truth, BPT 2048, 1.92h

(b) WPBGI 13.59m, 3.4GB

(c) DBPT, 256 14.66m

(d) SHPBGI, 256 coeff. 1.83h, 5.16GB

(e) SHPBGI, 64 coeff. 31.85m, 3.1GB

(f) PBGI 44.34m, 2.52GB

图 4.11 WPBGI 与 SHPBGI 进行对比。 (a) BPT， 每个像素 2048 个采样点，参考图象；(b) WPBGI，小波系数的个数为 $6 \times 32 \times 32$ (每个点/节点，进行非线性近似之前)，非线性近似的阈值为0.002； (c) 基于 SH 的 PBGI，64×3 个系数 (光泽反射的点/节点)； (d) 基于 SH 的 PBGI，256×3 个系数 (光泽反射的点/节点)； (e) PBGI 。 图像下面的数字表示总的渲染时间和内存使用峰值。

SHPBGI vs WPBGI 我们将 WPBGI 算法与 SHPBGI 算法的渲染结果在图 9 中进行对比，我们发现 SH 表示高频效果 (高频入射光线或者高频的 BRDF) 时需要大量的系数，即便是 256×3 也不能达到参照图效果。 WPBGI 虽然也需要大量的系数，但是支持非线性近似，从而只需要保存较少比例的系数。 此外，小波系数的层次编码进一步减少内存使用。

表 4.1 场景的性能和误差统计.

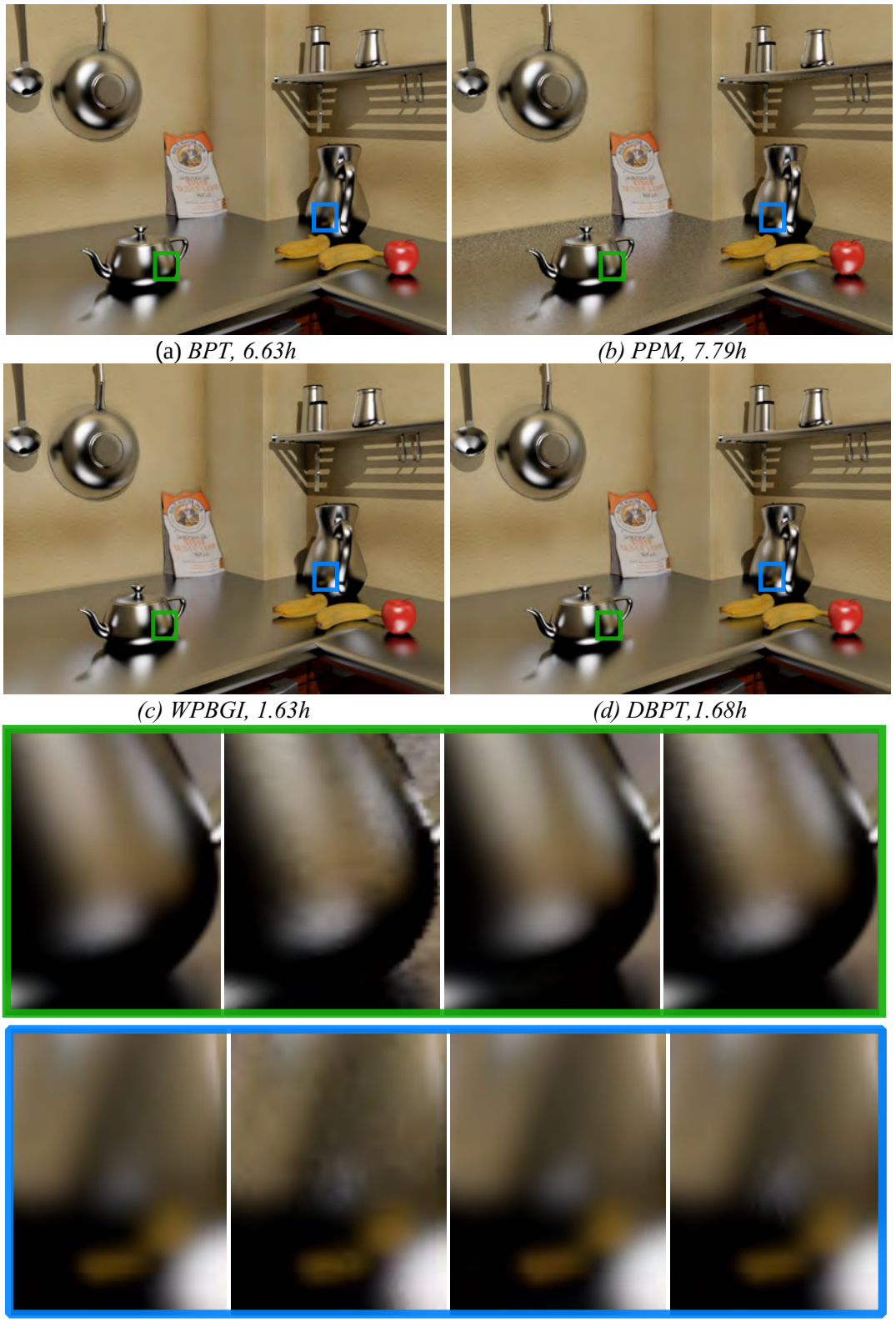
	BPT		WPBGI						Error
scene	num. samples	total (h)	pts. (M)	ω res	mem. (GB)	pr. (m)	ren. (h)	tot. (h)	*MSE*
Ring	16384	5.49	5	128^2	5.98	3.22	1.14	1.20	4.589e-5
Corner	16384	6.63	8	32^2	6.62	3.05	1.57	1.63	1.570e-4
Bunny	16384	10.53	4	32^2	5.68	12.28	0.84	1.04	6.527e-5
Kitchen	16384	11.20	5	32^2	5.26	6.57	3.66	3.77	7.948e-5
Sibenik	16384	10.33	10	32^2	5.85	11.07	1.75	1.94	2.099e-4

WPBGI vs 其他 在图 4.12 到图 4.15 中，将 WPBGI 与 BPT、DBPT 和 PPM 算法进行对比。 BPT 算法所使用的参数值为使结果没有明显噪声的最小采样数，在表格 4.1 中有详细的设置， PPM 与 BPT 有近似的渲染时间。 通过对比得出以下结论：与 BPT 相比，WPBGI 在保证较小的 MSE 且无走样的前提下，具有更高的效率； PPM 渲染结果有非常明显的噪声； DBPT 表示退化的 BPT，即使用较少的采样使得与 WPBGI 有相近的渲染时间，DBPT 渲染结果有大量的噪声。

图 4.16 和 4.17 中展示了使用 WPBGI 在不同 BRDF 光泽度下产生的渲染效果，并且与 BPT 算法进行对照。 通过对比，我们发现 WPBGI 算法能够产生与 BPT 相近的效果 (除了完全镜面反射之外)，并且具有更高的效率。

WPBGI 算法与 PBGI 算法一样，具有良好的时间连贯性。我们通过视频中的移动光源、移动相机以及变形的模型证明了这一点。

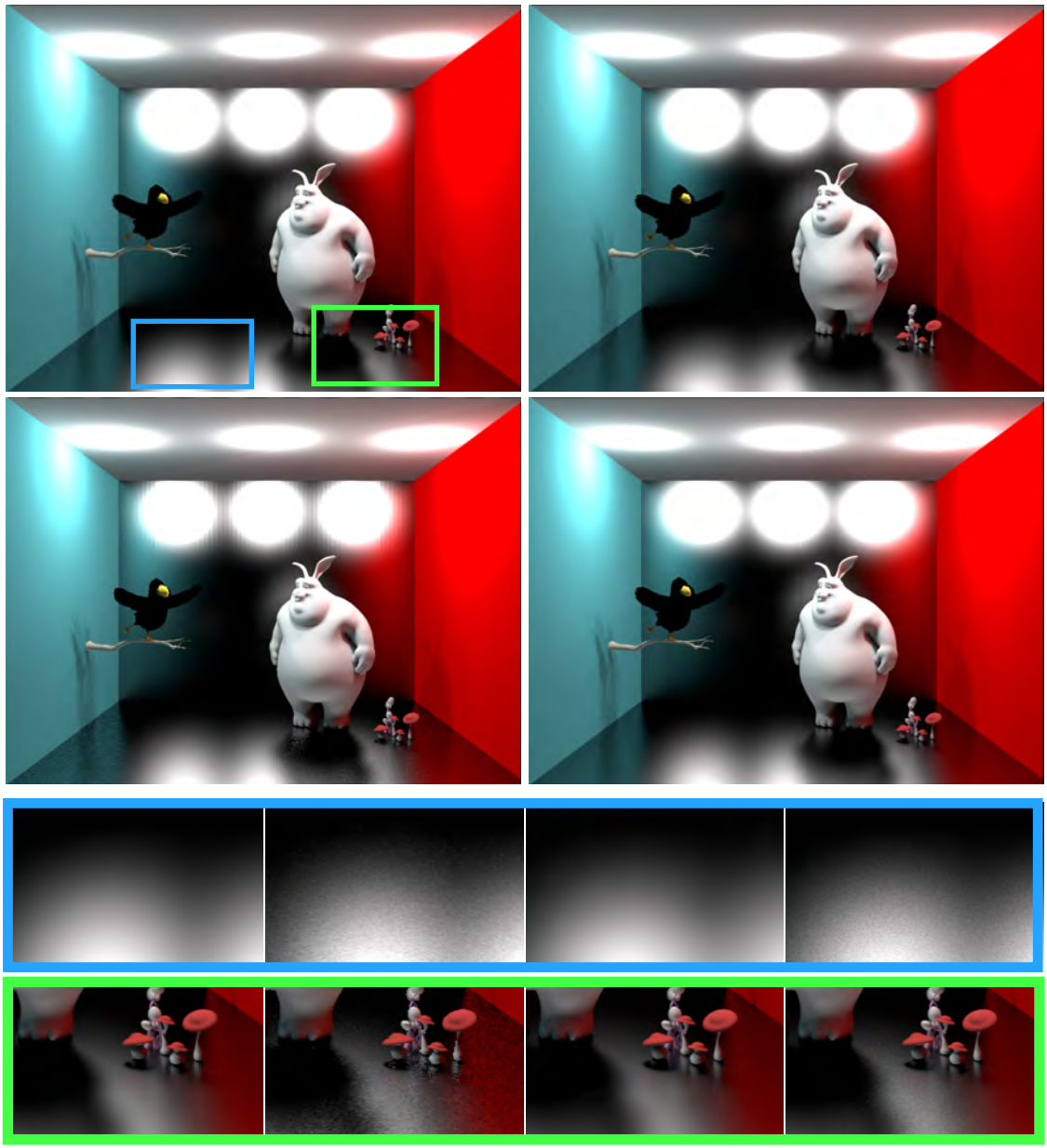
性能统计 我们在表 4.1 统计测试场景的渲染时间和误差情况，其中 *pts* 表示 WPBGI 算法点云中点的个数， ω *res* 表示每个点中小波系数的个数 (每个面)； *num.* 表示 BPT 算法中的采样数； *mem.* 表示在渲染过程中使用内存的峰值； *pr.* 表示点云产生和点云层次结构构建的时间； *ren.* 即着色点的渲染时间， 该时间包含了直接光照计算时间 (很短，可忽略) 和间接光照时间， 其中后者包括了微缓冲区初始化，点云层次结构遍历，点云层次结构节点投影以及微缓冲区与 BRDF 卷积所需要的时间； *tot.* 是以上两部分时间之和； 误差表示了 WPBGI 算法渲染结果与 BPT 算法渲染结果的均方差。 通过对照，可以得出如下结论： WPBGI 算法相对于 BPT 算法的加速比是 2.9x 到 10.1x，并且产生的误差是可忽略的。



(a) *BPT, 6.63h* (b) *PPM, 7.79h*

(c) WPBGI, 1.63h *(d) DBPT,1.68h*

图 4.12 Corner 场景对照。



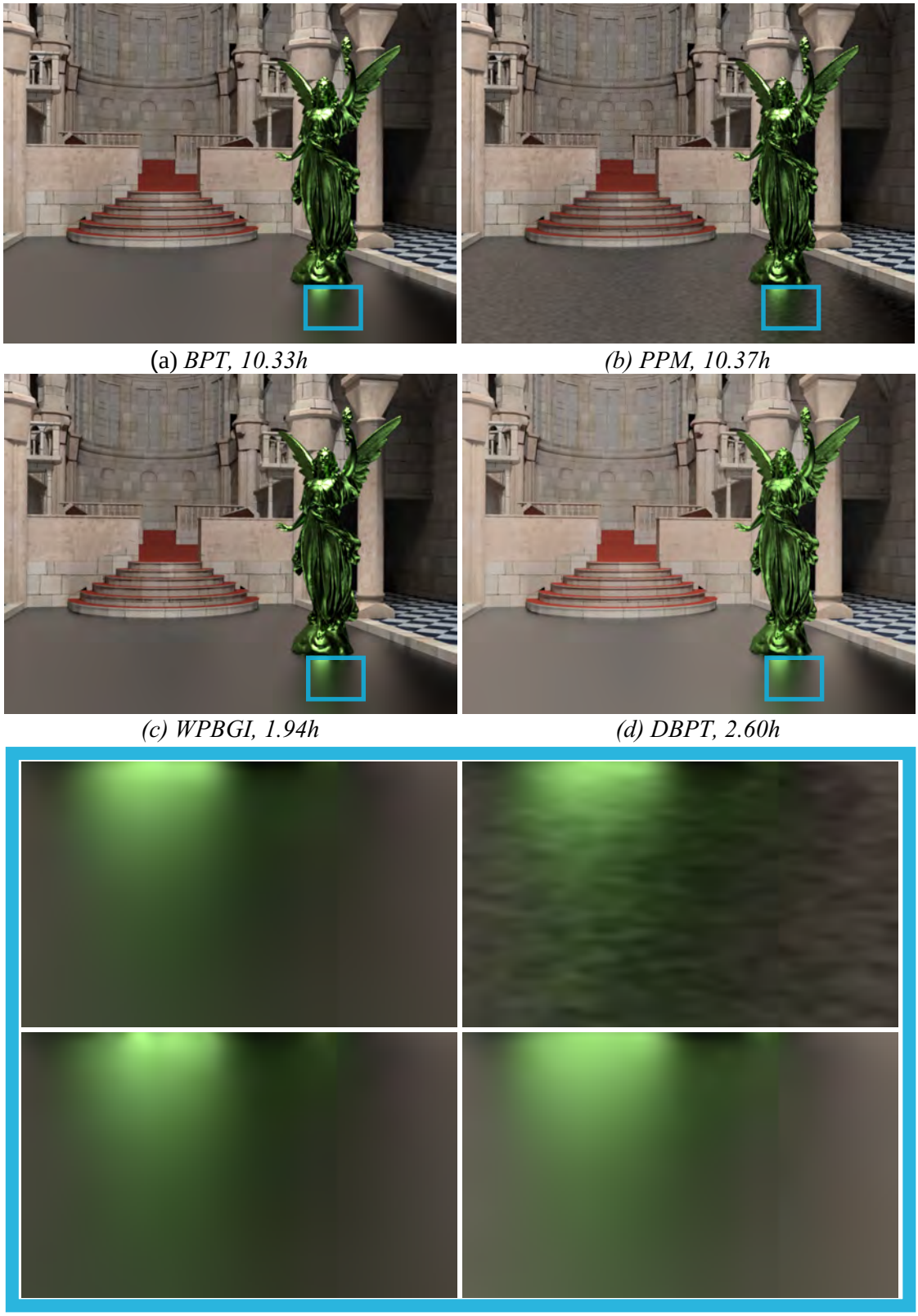
(a) *BPT, 10.53h* (b) *PPM, 10.07h* (c) *WPBGI, 1.04h* (d) *DBPT, 1.33h*

图 4.13 Bunny 场景对照

(a) *BPT, 11.20h*　　*(b) PPM, 11.55h*

(c) AWPBGI, 3.77h　　*(d) DBPT, 5.56h*

图 4.14 Kitchen 场景对照



(a) *BPT, 10.33h*　　*(b) PPM, 10.37h*

(c) WPBGI, 1.94h　　*(d) DBPT, 2.60h*

图 **4.15** Sibenik 场景对照

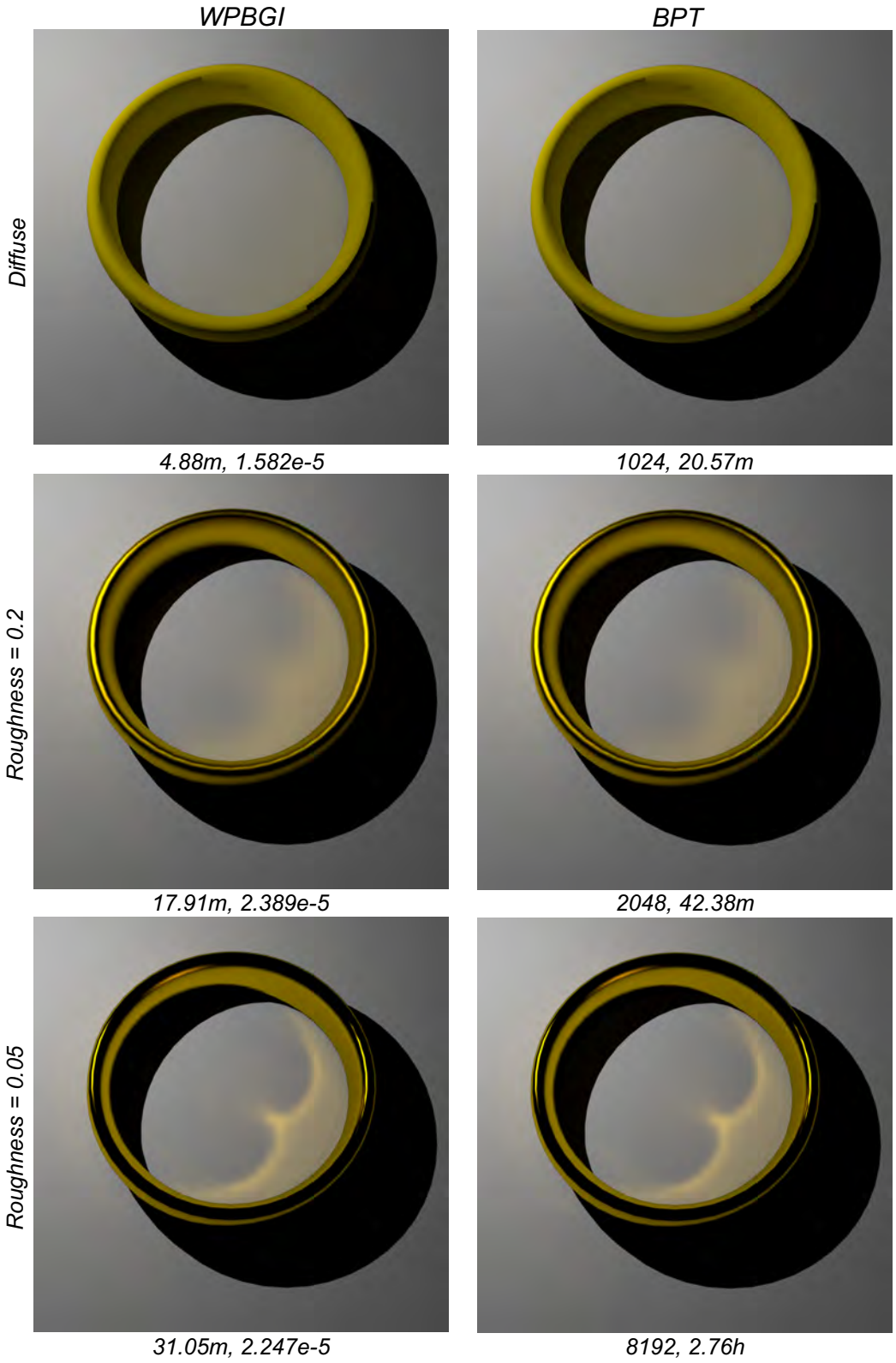


图 4.16 Ring 场景在不同反射程度下 (完全漫反射、0.2、0.05) 的渲染结果。

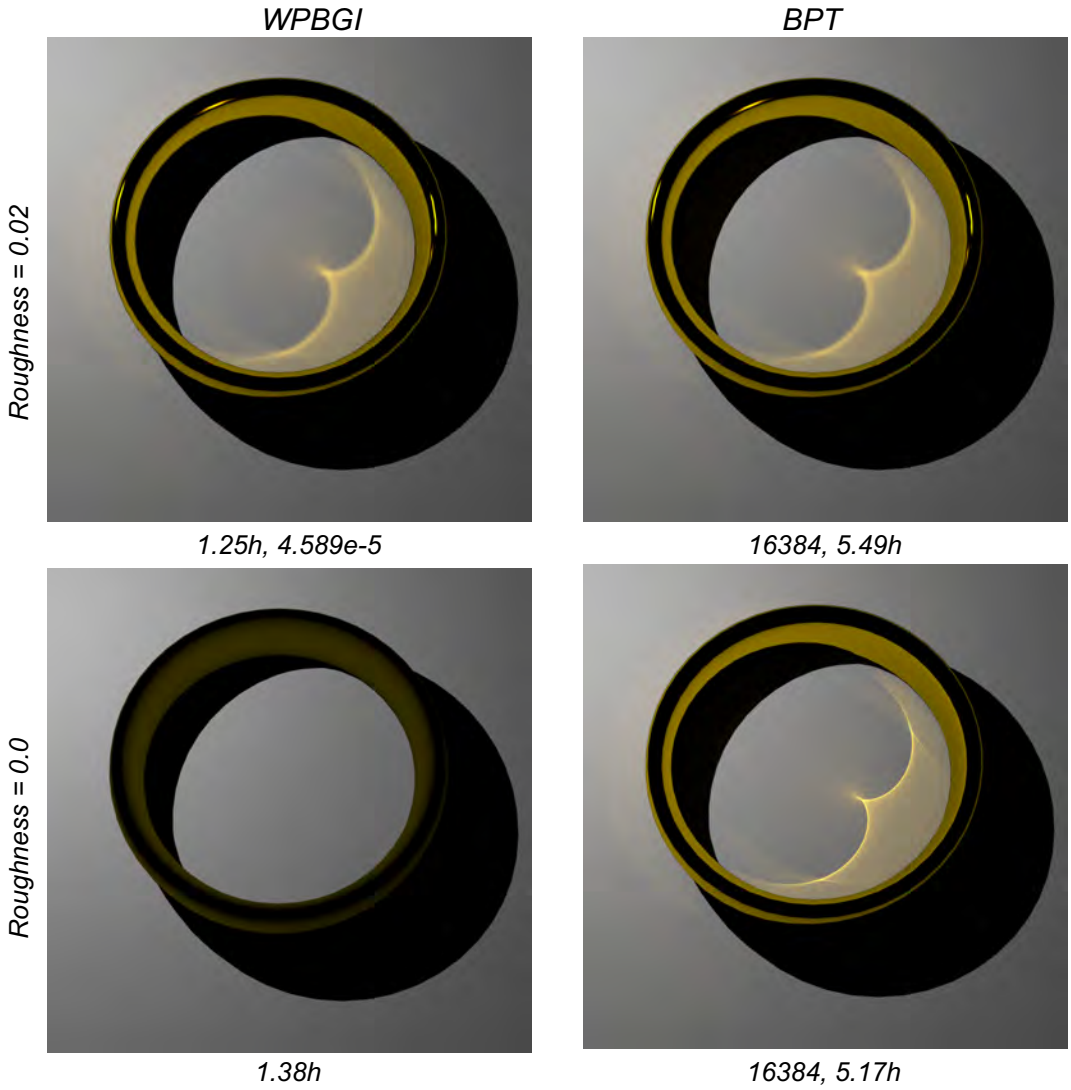


图 4.17 Ring 场景在不同反射程度下 (0.02、0.0) 的渲染结果。 左图数字表示总渲染时间和误差 (*MSE*)，右图数字表示采样点个数和总渲染时间。

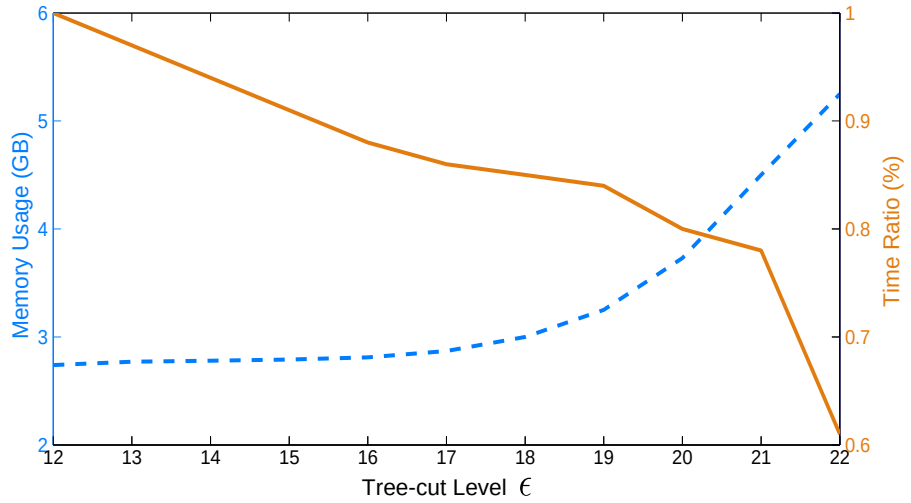


图 4.18 对于不同的层 ϵ，内存使用(蓝色)和渲染时间比率(都除以层12 的时间)情况统计。

4.7.2 主要参数分析

图 17 统计了 Corner 场景 (使用 5M 点云) 在不同的划分层 (ϵ) 情况下，点云层次结构的内存使用情况。随着 ϵ 的减少，小波系数层次化编码技术能够有效地降低内存使用直到趋于平滑。 趋于平滑的原因在于，当节点的层次逐渐降低时，节点与其邻居节点的辐射亮度相似度降低，以至系数向量相似度降低，因此不能产生值较小的节点细节系数，从而在使用基于阈值的非线性近似时被删除的系数减少。 在减少层数时，由于重构节点近似系数时所需要的时间更多，因此需要更多的渲染时间。 通过衡量，我们认为 $\epsilon = 20$ 是一个适合的值。

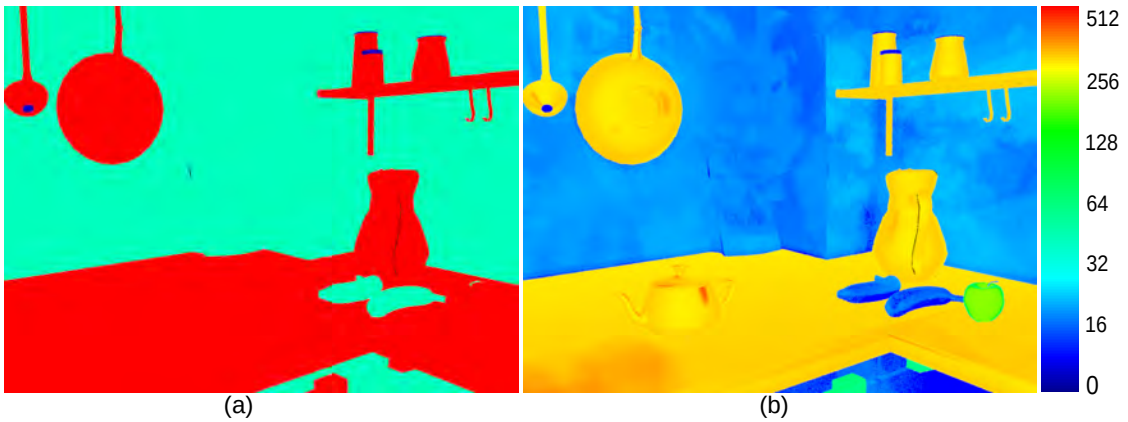


图 4.19 对使用了 AMB 的 WPBGI 和没有使用 AMB 的 WPBGI 算法的微缓冲区的平均分辨率进行对比。

图 18 对比了采用重要性驱动微缓冲区前后所使用的平均分辨率的值。 对于图中所示的场景，在这两种情况下，所需要总的渲染时间分别是大于 24 小时和 1.63 小时。 由此得出结论，重要性驱动微缓冲区技术能够大幅度地提升速度。

4.8 总结

首先我们讨论 WPBGI 算法的不足之处。 第一，用于进行直接光照采样的立方体贴图像素大小或者说表示辐射亮度的小波系数个数(压缩前)是由用户输入的，而不是自适应的，影响算法的易操作性。 第二，用于光照重要性驱动微缓冲区的度量，我们采用入射光线辐射亮度值，实际上所需要的度量应该能够抽象高频特征， 而辐射亮度的值大并不能代表入射光线是高频的， 对于其他的度量，比如一个节点的辐射亮度的平均差，或者表示入射辐射亮度小波系数的层等等，我们也进行了实验，却不能得到比使用辐射亮度值更好的效果。 第三，当投影到微缓冲区某像素的节点 n 满足光照重要性驱动函数时，该像素里面存储的所有的节点都要被重新遍历和投影， 实际上 n 可能被其他节点阻挡，在这种情况下提高微缓冲区的分辨率对于提升质量没有任何帮助，但是却影响了效率， 如果只对距离着色点最近的节点进行光照重要性判断是不充分的，即很多高频情况无法识别出来。

在本章中，我们提出了基于小波的点缓存全局光照算法，对点缓存全局光照算法进行扩展，从而支持非漫反射光线传递的计算。 此外，我们提出了小波系数的层次化编码技术，从而有效地减少内存使用。 最后，为了解决高频入射光线或者高频 BRDF 中出现的走样问题，我们提出了基于重要性驱动的微缓冲区构建方法。 最终，我们的算法相对于 BPT 算法，有 3 到 10 倍提速，易于在当前的 PBGI 框架中实现，并且只有较少的可控参数。

关于本文的未来工作，有以下几点。 首先，我们的算法中目前只支持一次反弹的光泽反射，将其扩展到多次反弹并且支持折射会产生更加有趣的效果； 其次，当出现高频 BRDF 时，对点云的密度要求较高，继续采用规则的采样，将产生巨大的内存压力，因此如何更加灵活的采样也将是研究的关键问题。 最后，我们使用了全局坐标系下的立方体贴图上的 2D 哈尔小波来表示点的出射辐射亮度，而微缓冲区使用的局部坐标系下的半球空间， 当扩展到多次反射时，需要大量的坐标系变换，更好的解决办法是采用 [66] 中的球形小波。

第 五 章　PBGI 的多次反射快速计算方法

PBGI [24] 中，在需要计算多次反射时采用如下方法：将点云中的每个点当作着色点，遍历点云层次结构找到树切，将树切投影到微缓冲区，微缓冲区与 BRDF 进行卷积，从而计算出每个点的间接光照，即一次反射光照，重复该操作即可得到每个点多次反射的光照。由于 PBGI 只支持漫反射光线的传递，计算得到的 n 次反射光照与其中存储的直接光照相同可以使用 RGB 表示 (节点中使用 SH 表示)。

第四章提出了基于小波的 PBGI (WPBGI) 算法计算非漫反射光线的传递，本章将在 WPBGI 基础上提出快速计算多次反射的模型。考虑到点云中点的空间连贯性，我们提出视点树 *(View Tree)* 的概念，即点云树对自身 (视点树) 进行遍历和投影，而不是对点云中的每个点，从而提高遍历效率，该方法同时也是对第三章中基于分解的 PBGI 算法的推广，将聚类单层的结构推广到树多层的结构，增加重用的自适应性，消除对聚类的依赖。我们提出了从入射辐射亮度和 BRDF 快速计算出射辐射亮度的模型，从而进一步提高多次反射光照计算效率，与 BPT 算法相比，我们的多次反射算法具有 9 到 17 倍左右的提速。此外，我们将球形微缓冲区代替半球微缓冲区来模拟折射材质下的光线传递。

最后，我们利用 GPU 的并行计算能力，将存储了多次反射的点云层次结构用于预览渲染，支持视点改变、漫反射以及各种频率的光泽反射材质的渲染。

5.1 相关工作

在预览渲染阶段，我们采用基于 GPU 的渲染技术用于加速渲染， GPU 由于其强大的并行计算能力，在渲染上体现出很大的优势，并且得到了广泛的应用，本节中将介绍相关技术。

5.1.1 延迟着色

光栅化 (Rasterization) 是将几何数据经过一系列变换之后最终转化为像素， 从而呈现在显示设备上的过程，此外，光栅化也可以看作是 3D 的几何数据投影到缓冲区上的过程。 场景中可能存在大量的几何数据，而每个物体的投影操作是相

对独立的， 图像处理单元 (Graphics Processing Unit，简称 GPU) 在光栅化方面具有很大的优势， 因此很多实时或者可交互的全局光照应用使用到光栅化技术，比如延期着色 (Deferred Shading) [67]， 该技术将位置、法向、着色属性 (比如漫反射颜色) 等通过光栅化存到缓冲区中， 然后将这些缓冲区用于辅助光照或者其他计算，该技术能够有效地提高渲染效率， 广泛应用到实时或者可交互的全局光照计算中 [68] [69] [70] [71] [72]，本章也采用了该技术。

5.1.2 屏幕空间投影技术

在多光源算法以及辐射亮度缓存 (Radiance Caching) 中， 三维空间中分布着表示间接光照的虚拟光源或者采样点 (统称为采样点)，用于计算每个着色点的全局光照， 有两种计算方式：聚集 (gathering) 和投影 (splatting)， 前者是针对每个着色点进行的操作，而后者则是从采样点作为出发点。 后者与前者相比，优势在于 GPU 可以高效地将每个采样点分别向屏幕空间进行投影， 而前者则需要将采样点组织到树 (光源割中的树，辐射亮度缓存中的树) 中，然后进行遍历 (查找近邻)，而 GPU 并不能高效地操作层次结构， 因此投影技术得到广泛应用。 Gautron 等人在 [73] 中提出适用于 GPU 的辐射亮度投影技术来计算采样点对着色点的贡献，在采样点投影阶段，将每个采样点作为一个球向屏幕空间投影，该采样点在图像上将覆盖一定范围的像素 (着色点)，根据如下公式计算出权重：

$$w_k(p) = \frac{1}{\frac{\|p-p_k\|}{R_k} + \sqrt{1 - n \cdot n_k}}. \tag{5.1}$$

其中 p 和 n 表示着色点的位置和法向， p_k 和 n_k 表示采样点的位置和法向，R_k 是从 p_k 可见物体的调和平均距离。 如果该权重大于预设值，则根据采样点存储的入射辐射度和梯度计算入射辐射亮度， 与 BRDF 卷积后得到着色点的出射辐射亮度，将该辐射亮度和权重分别累加到辐射亮度缓冲区和权重缓冲区中。 在渲染阶段，每个着色点从辐射亮度缓冲区和权重缓冲区中得到辐射亮度累加和以及权重累加和， 前者除以后者之后得到该着色点的间接光照。 该算法相比辐射照度缓存或者辐射亮度缓存有大幅度效率的提升，该技术也应用于本章中。 Dachsbacher 等人在 [69] 中利用投影来计算二级光源 (secondary lights) 对着色点的辐射亮度，每个二级光源投影为一个四边形，四边形的大小根据二级光源距离屏幕的距离等信

息来确定，最终使得该算法可以实时地计算漫反射或者非漫反射的间接光照，并且支持光源的变化。 [70] 和 [71] 中根据根据二级光源所在表面材质的不同，按照不同的投影大小投影到多分辨率缓冲区中。

5.1.3 预卷积技术

在计算出射辐射亮度时，需要将入射辐射亮度与该点的 BRDF 进行卷积， 在实时渲染应用中，与高频非漫反射 BRDF 卷积很耗时， 因此在预处理时先将两者进行预卷积则可以避免在运行时刻 (run-time) 进行计算， 比如在 [74][75] 以及 [76] 中都采用了预卷积技术。

第三章中介绍了辐射照度缓存 (IRC) 和辐射亮度缓存 (RC) 技术，在 IRC 中，利用采样点缓存的辐射照度 (RGB) 插值后与漫反射表面的 BRDF 相乘得到着色点的辐射照度； 在 RC 中，将采样点缓存的辐射亮度 (SH 系数) 插值后与表示 BRDF 的 SH 系数相乘得到着色点的出射辐射亮度。 在后者中，每个着色点的辐射亮度系数与 BRDF 系数进行卷积操作的时间复杂性与 SH 系数个数成线性正比， 于是系数较多时卷积操作很耗时。Scherzer 等人在 [76] 中提出预卷积辐射亮度缓存 (Pre-convolved Radiance Caching) 技术，将每个采样点缓存中的入射辐射亮度与 BRDF 进行预卷积，将漫反射部分存入到低分辨率的图片中，将镜面反射部分根据材质光泽度的不同存入到不同的 mipmaps 层次中；在渲染阶段，每个着色点只需要到两部分缓存中查找即可，该方法的结果与 RC 有相近的效果，而与 IRC 有相近的效率。 本文中为了避免在运行时进行卷积，使用小波系数存储节点的出射辐射亮度，然后用于预览渲染。

5.2 算法概述

在本章中，我们提出了 PBGI 的多次反射 (折射) 的快速计算模型 (view tree base PBGI，简称为 VPBGI)，主要包括了基于视点树的遍历和投影技术以及出射辐射亮度快速计算技术，算法的流程如下：

1. 预计算表示 BRDF 的小波系数；
2. 基于视点树对点云树进行遍历，计算入射辐射亮度；
3. 计算出射辐射亮度，并且更新点云树。

多次迭代该过程，则在点云树中存储了直接光照和多次反射的间接光照，最后将该点云层次结构用于着色点的高质量的全局光照计算或者预览渲染。图 6 中描述了改进前和改进后的多次反射计算过程。

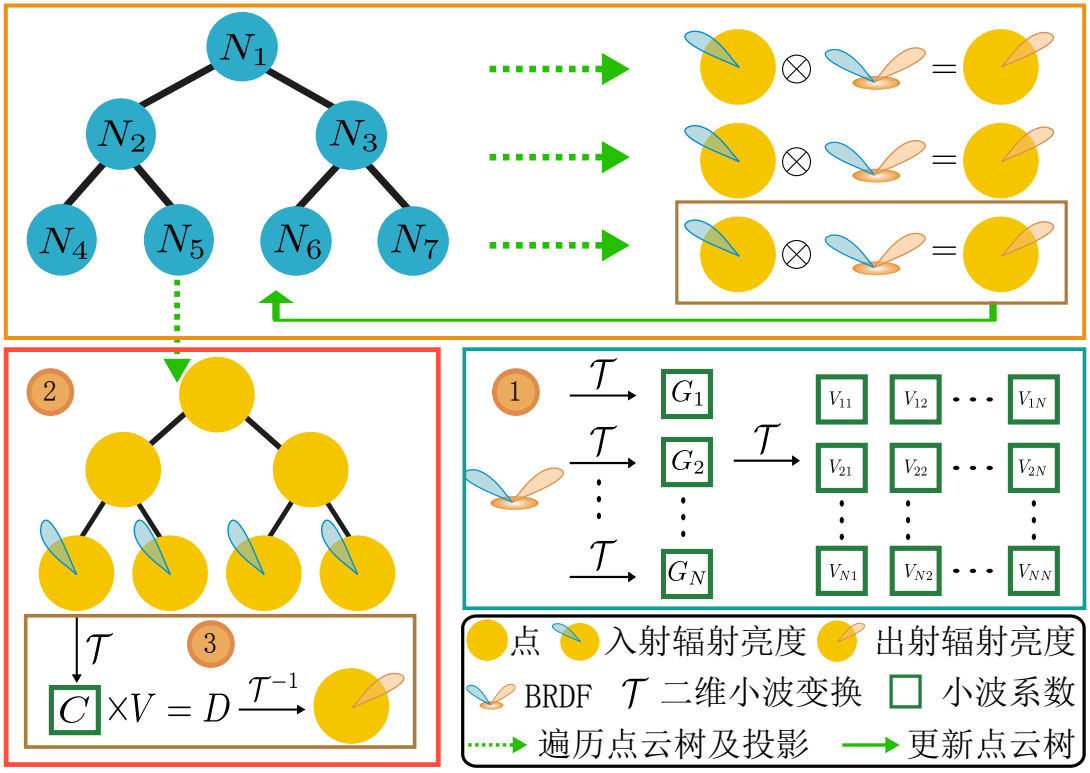


图 5.1 橙色矩形中表示了改进前的多次反射计算流程：每个点 (橙色圆圈) 对点云树 (蓝色) 进行遍历并且将节点投影得到每个点的入射辐射亮度，该入射辐射亮度分别与各个出射方向的 BRDF 卷积得到出射辐射亮度，按照点云树中的表示形式 (立方体贴图上的二维小波) 进行变换，并且更新到点云树中。我们改进后的算法是：进行 BRDF 的预计算得到四维小波系数 (蓝色矩形)；使用视点树 (黄色) 来遍历点云树，计算出入射辐射亮度，然后进行小波变换 $\mathcal{T}$；利用出射辐射亮度计算模型计算出射辐射亮度，并且更新点云树。

5.3 多次反射计算模型

在本节中，我们将详细介绍快速多次反射计算模型。

5.3.1 BRDF 表示

在 PBGI 算法中，入射辐射亮度与 BRDF 卷积后得到出射辐射亮度，表示为：

$$L_o = L_i \otimes \rho.$$

其中 L_o 为出射辐射亮度， L_i 为入射辐射亮度 (已包含 cosine 项)， ρ 为 BRDF。对着色点计算出射辐射亮度时，由于出射方向是确定的，时间复杂性为 $O(N)$，其中 N 表示微缓冲区像素的个数或者入射方向的采样数。在计算多次反射时，需要计算非漫反射点在各个出射方向的辐照亮度，如果在每个出射方向，分别进行以上卷积操作，则需要的时间复杂性为 $O(N^2)$，假设 N 也为出射方向的采样个数。在高频 BRDF 情况下，需要较高的采样数，以上计算很费时。

已知空间域中的卷积操作等同于频率域 (比如小波) 中的乘法操作，而小波系数具有稀疏性的特点，在频率域中计算出射辐射亮度将会提高效率，因此，我们将入射光照和 BRDF 分别使用小波系数表示。

BRDF 在局部坐标系 (z 轴与表面法向一致) 下可以参数化为 $\rho(\omega_i, \omega_o)$，是 4D 函数，而在全局坐标系下，还要考虑到法向，即函数增加了两维，考虑到存储问题，最终采用局部坐标系来表示 BRDF。

关于如何使用小波系数表示该函数，有两种方案：一种方案是对每个 ω_o，把关于 ω_i 的二维函数使用小波系数表示；另外一种方案，将 ρ 进行四维小波变换。以下将分别介绍和分析这两种方案。

第一种方案中，在每个 ω_o 方向 (比如分辨率为 32×32) 上，对半球上的每个 ω_i 进行采样，则每个 ω_o 对应一个图像，对每个图像进行二维小波分析并且进行非线性近似，最后得到表示 BRDF 的矩阵 M ，表示如下：

$$M = \mathcal{T}\rho = [G_1, G_2, ..., G_i, ..., G_N]^T .$$

其中 $\mathcal{T}$ 表示二维小波变换 (在 ω_i 上)，G_i 表示了行 i 在对应出射方向下的小波系数。令入射辐射亮度进行二维小波分析得到入射辐射亮度系数表示为 C，通过以下公式计算出射辐射亮度 L_o：

$$L_o = C \times M.$$

即 C 与 M 中每行表示某出射方向光照的系数相乘得到在该方向的出射辐射亮度，从而得到整个半球方向上的出射辐射亮度。

第二种方案中，对 $\rho(\omega_i, \omega_o)$ 进行四维小波变换，首先在 ω_i 上进行二维小波变换，然后，再在 ω_o 上进行二维小波变换 (如图 6 所示)，表示如下：

$$V = \mathcal{T}\mathcal{T}\rho = \mathcal{T}[G_1, G_2, ..., G_i, ..., G_N]^T .$$

在计算出射辐射亮度时，有以下公式成立：

$$D = C \times V, \quad L_o = \mathcal{T}^{-1} D. \tag{5.2}$$

其中，D 表示表示入射光照的二维小波系数 C 与表示 BRDF 的四维小波系数 V 相乘的结果，该结果经过二维小波逆变换之后得到出射辐照亮度。

第二种方案与第一种相比，所产生的结果具有更大的稀疏性，需要的存储空间更少，逆变换产生的额外开销较小，在效率上比第一种方案有很大提高，因此，我们采用第二种方案。

5.3.2 基于视点树的遍历策略

第三章中提出了 FPBGI，聚类内的着色点重用树切，在本章中，我们提出把着色点组织到树中来重用树切，从而使得重用具有自适应性，该树称为视点树。

在多次反射计算中，点云中每个点需要像着色点一样计算间接光照，因此点云层次结构本身就是视点树，为了表达清晰，我们使用两个不同的名称：视点树和点云树。我们将视点树中的节点称为接收节点，点云树中的节点称为发射节点。采用视点树的原因在于，我们发现某些发射节点对某些接收节点中的所有点具有类似的贡献，令 N_i/V_j 表示发射节点接收节点对，简称节点对，其中 N_i 表示发射节点，V_j 表示接收节点，进行遍历的过程就是找N_i/V_j对的过程。着色点遍历点云树时，发射节点是否需要细分由该节点与着色点之间立体角决定，该立体角称为发射立体角。同样地，我们使用接收节点与发射节点中心之间的立体角来决定接收节点是否需要细分，称为接收立体角。

遍历过程从视点树的根节点和点云树的根节点产生的节点对开始，如图 5.2 所示。对每个N_i/V_j对进行如下操作：如果接收节点在发射节点内部，将该发射节点分别与接收节点的子节点形成节点对。如果接收立体角大于阈值，说明发射节点无法被该发射节点中的所有点重用，因此将该发射节点分别与接收节点的子节点形成节点对。如果接收立体角小于等于阈值，说明该发射节点可以被接收节点中的所有节点重用，那么对该接收节点遍历该发射节点，找到满足发射立体角的发射节点后，将每个发射节点以及计算得到的方向和距离信息存入到该接收节点中，对该接收节点构建微缓冲区 (其法向作为 z 轴正方向)，并且将缓存在该接收节点中的发射节点在该微缓冲区中投影。如果当前接收节点没有发射节点与之形

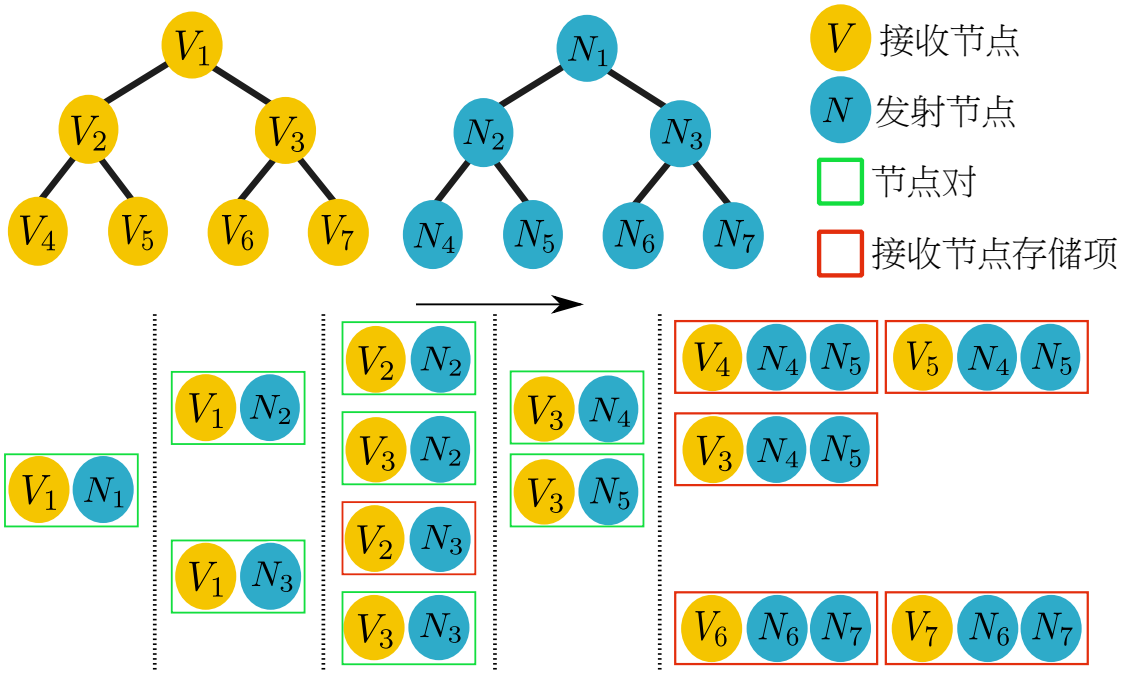


图 5.2 视点树 (黄色) 对点云树 (蓝色) 的遍历过程。遍历过程从两棵树的根节点开始，由于发射节点 (N_1) 包含接收节点 (V_1)，N_1 被细分，从而形成节点对 V_1/N_2 和 V_1/N_3，由于 V_1 发射节点立体角过大，V_1 被细分，分别形成 V_2/N_2、V_2/N_3、V_3/N_2 和 V_3/N_3。V_2/N_3 节点对中，由于同时满足发射立体角条件和接收立体角条件，将 N_3 项存储在 V_2 中，而 V_3/N_2 满足发射节点立体角条件，但未满足接收节点立体角条件，因此细分为 V_3/N_4 和 V_3/N_5，由于 N_4 和 N_5 分别是叶节点因此两者被存储在 V_3 中。其他节点对依次处理。

成为节点对，那么开始处理其子节点。以上的操作一直进行到接收节点为叶节点为止。

接收中间节点遍历点云树时，根据该节点和发射节点的法向和位置信息来判断发射节点是否在接收节点的视角范围内，如图 5.3 所示。如果未在视角范围内，那么该发射节点无需继续进行处理，直接被忽略。

每个接收叶节点的树切分布在自己存储的节点列表以及从视点树根节点到该节点路径上的接收节点中。在投影这些节点之前，首先为该接收叶节点构建微缓冲区。第四章中提出的自适应微缓冲区技术的前提是着色点的出射光线方向是判断，在本章中，虽然把点云中的每个点当作着色点来计算间接光照，但是该点是视点独立的，因此点的微缓冲区构建不能使用 BRDF 重要性驱动来构建，但是可

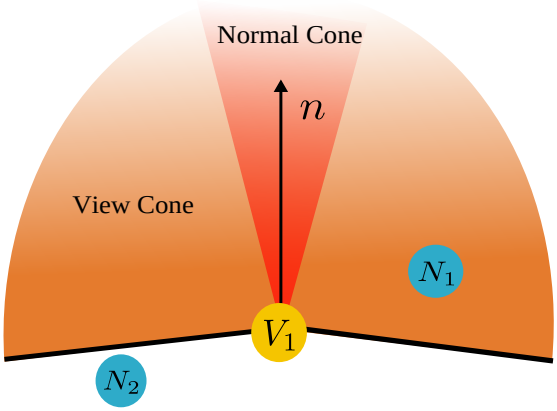


图 5.3 接收节点 V_1 的法向以及法向范围角 (normal cone，红色)，以及产生的视角范围 (view cone，橙色)。在视点范围外的发射节点 N_2 不再处理。

以使用入射光重要性驱动。构建完微缓冲区之后，处理其树切中的节点，按照与该接收叶节点距离 (第三章中介绍了该概念)，先处理自己列表中的发射节点，然后从近到远分别对每个接收中间节点的存储项进行处理，如果当前接收叶节点与中间节点的法向相似，那么直接重用其微缓冲区，而不用对每个存储项进行投影，反之则将存储项中的发射节点向该接收叶节点的微缓冲区进行投影。当处理完路径中所有节点时，得到了入射辐射亮度。

5.3.3 出射辐射亮度计算

每个接收叶节点计算得到入射辐射亮度之后，将其进行二维小波变换从而得到小波系数，BRDF 也用小波系数表示。这两个系数都进行非线性近似，并且使用树作为数据结构来存储，两者都是稀疏的。通过公式 3 计算出射辐射亮度系数。WPBGI 中使用全局坐标系下的立方体贴图对出射辐射亮度采样后分别对各面进行小波变换，而现在得到的系数是局部坐标系中的表示，由于小波系数不易进行旋转，因此首先将局部坐标系下的出射辐射亮度系数进行逆变换，即从频域变换到空间域，在空间域中，按照全局坐标系下的立方体贴图进行采样，然后分别对每个面进行二维小波变换从而回到频域。

5.3.4 更新小波树

在计算出接收叶节点的出射辐射亮度小波系数之后，要将其更新到整个点云树 (小波树) 中，与构建小波树时相同，后续遍历小波树，计算每个节点的间接出

射辐射亮度。在更新过程中，依然采用小波层次编码技术，按照每个节点中存储的小波系数的类型分别进行更新。在计算出所有节点的间接出射辐射亮度之后，与表示直接辐射亮度的小波系数相加，从而使得点云树中存储了直接光照和一次间接光照，如果需要计算更多次反射，则将本次计算得到的间接光照分布在小波树中，重复进行上述操作。

5.3.5 计算折射

在多次反射基础上，我们将应用推广到折射计算。算法中主要的不同之处有以下几点：之前采用半球作为微缓冲区，当推广到折射之后，使用球形微缓冲区；对入射光照进行小波变换时，分别对两个半球进行二维小波变换；在表示 BSDF 时，对局部坐标系下产生的两个半球分别进行二维小波变换；此外，在计算树切时，之前采用的节点是否在法向视角内的判断也需要进行修改。

5.4 预览渲染

当多次反射的辐射亮度分布在点云树中后，在本节中利用该层次结构，经过 GPU 加速后实现场景的预览渲染，场景中的材质支持漫反射以及各个频率的光泽反射，并且支持视点的变化。

5.4.1 算法

点云树中用小波系数存储了出射辐射亮度，其中的 ϵ 层中存储了节点近似系数和节点细节系数，我们利用该层中的所有节点作为采样点来插值产生所有着色点的辐射亮度。为了计算每个采样点对着色点的权重，我们使用 [73] 中提出的辐射亮度缓存投影技术。预览渲染算法主要分为三步 (如图 5.4 所示)：

1. 几何阶段：将当前场景进行延迟着色 (Deferred Shading)，分别产生位置缓冲区、法向缓冲区等 G Buffer。
2. 节点阶段：将点云树 ϵ 层的节点进行辐射亮度投影，根据公式 5.1 计算当前节点对所覆盖像素中着色点 (通过 G Buffer 获得) 产生的权重，并且将权重与该着色点的辐射亮度相乘后累加到加权辐射亮度累积缓冲区，而权重则累加到权重缓冲区。点云树是视点独立的，当改变视点时，根据出射方向将小波系数重构出辐射亮度即可。

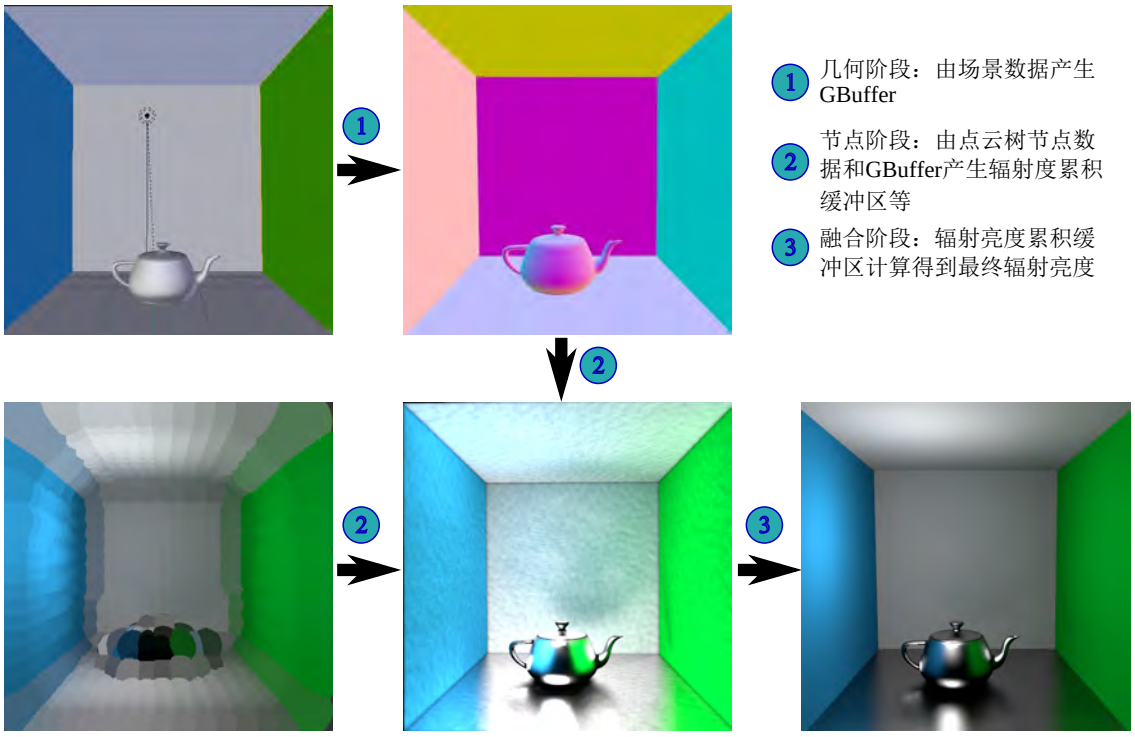


图 5.4 预览渲染流程图

3. 融合阶段：进行当前场景着色，通过辐射亮度累积缓冲区和权重缓冲区查找获取到当前着色点的辐射亮度累加和以及权重累加和，将前者除以后者后得到该着色点的间接光照，与直接光照相加后得到着色点的全局光照。

5.4.2 实现细节

我们将所有节点的小波系数存储在一个一维纹理缓冲区 (texture buffer) 中，在每个节点中存储小波系数根节点的索引，此外，节点的其他数据 (比如节点的半径) 也使用 texture buffer 进行存储。

在第二个阶段中，每个节点作为一个 vertex 执行 vertex shader，在 geometry shader 中根据节点半径，将每个点构造边长为 $2r$ 的正方形 (r 表示节点的半径)，之后在 fragment shader 中进行投影。

5.5 结果及讨论

我们的技术在 Mitsuba Renderer [2] 上实现， 实验的硬件环境是 2.67 GHz Intel i7 (8核) 处理器，9 GB 内存，GTX 480 2GB显存， OpenGL 的版本为 4.3， 离线渲

表 5.1 场景的性能和误差统计.

	BPT		VPBGI					Error
scene	num. samples	total (h)	pts. (M)	mem. (GB)	pr. (m)	ren. (m)	tot. (h)	*MSE*
Ring	16384	6.16	4	4.91	11	30.90	0.70	2.311e-5
Teapot	16384	18.33	4	5.01	23.63	58.00	1.36	1.163e-4
Sphere	16384	17.5	2	3.21	10.66	51.14	1.03	1.361e-4
Torus	65536	40.01	1	6.53	201.61	60.06	4.37	3.116e-4

表 5.2 预处理时间对比.

			PBGI			VPBGI		
scene samples	bounces.	tree constr. (m)	travers. (m)	rad. (m)	tot. (m)	travers. (m)	rad. (m)	tot. (m)
Ring	2	2	12.60	2.19	14.77	6.47	2.52	8.99
Ring	3	2	15.62	1.99	17.61	9.37	2.05	11.42
Teapot	2	4	43.37	4.63	48.00	15.13	4.50	19.63
Teapot	3	4	53.7	5.30	59.00	21.19	5.81	27.00
Sphere	2	0.69	22.63	3.39	26.02	6.39	4.03	10.42
Sphere	3	0.69	26.24	3.94	30.18	11.36	4.54	15.90

染结果的分辨率为 1024×1024 (Torus 除外，1024×768)，并且使用 32×32 分块，自适应采样用来反走样，预览渲染结果的分辨率为 512×512，同 BPT 算法进行对照，在所有的对照中，使用均方差 (MSE) 来衡量误差。在渲染过程中，使用 PBGI 技术计算间接光照，而直接光照则采用光线跟踪进行计算。

正确性验证 在图 5.5、图 16，图 14 和图 15 中，将 VPBGI 与 BPT 算法进行对比，BPT 算法所使用的参数值为使结果没有明显噪声的最小采样数，在表格 3 中有详细的设置。通过对比得出以下结论：与 BPT 相比 VPBGI 在保证较小的 MSE 且无走样的前提下，具有更高的效率，大约为 9x 到 17x。

在图 5.6、图 5.8 和图 5.10 中，我们将具有相同渲染时间的各个算法进行对照。第一行的 BPT 算法作为对照；DBPT 表示退化的 BPT，即使用较少的采样使得与 VPBGI 有相近的渲染时间，DBPT 渲染的效果有大量的噪声；PPM 渲染的结果也存在大量的噪声。

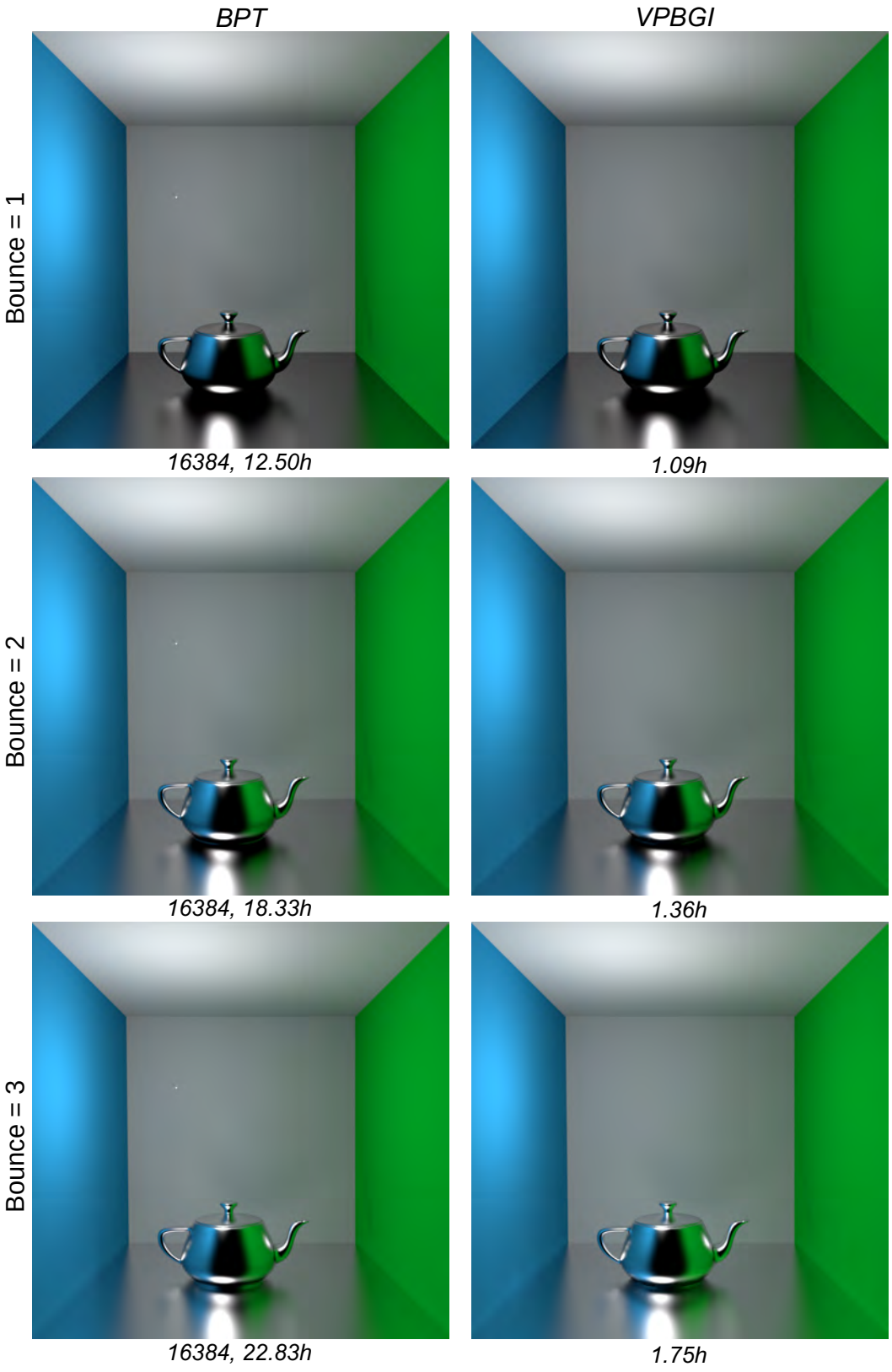


图 **5.5** Teapot 场景对照

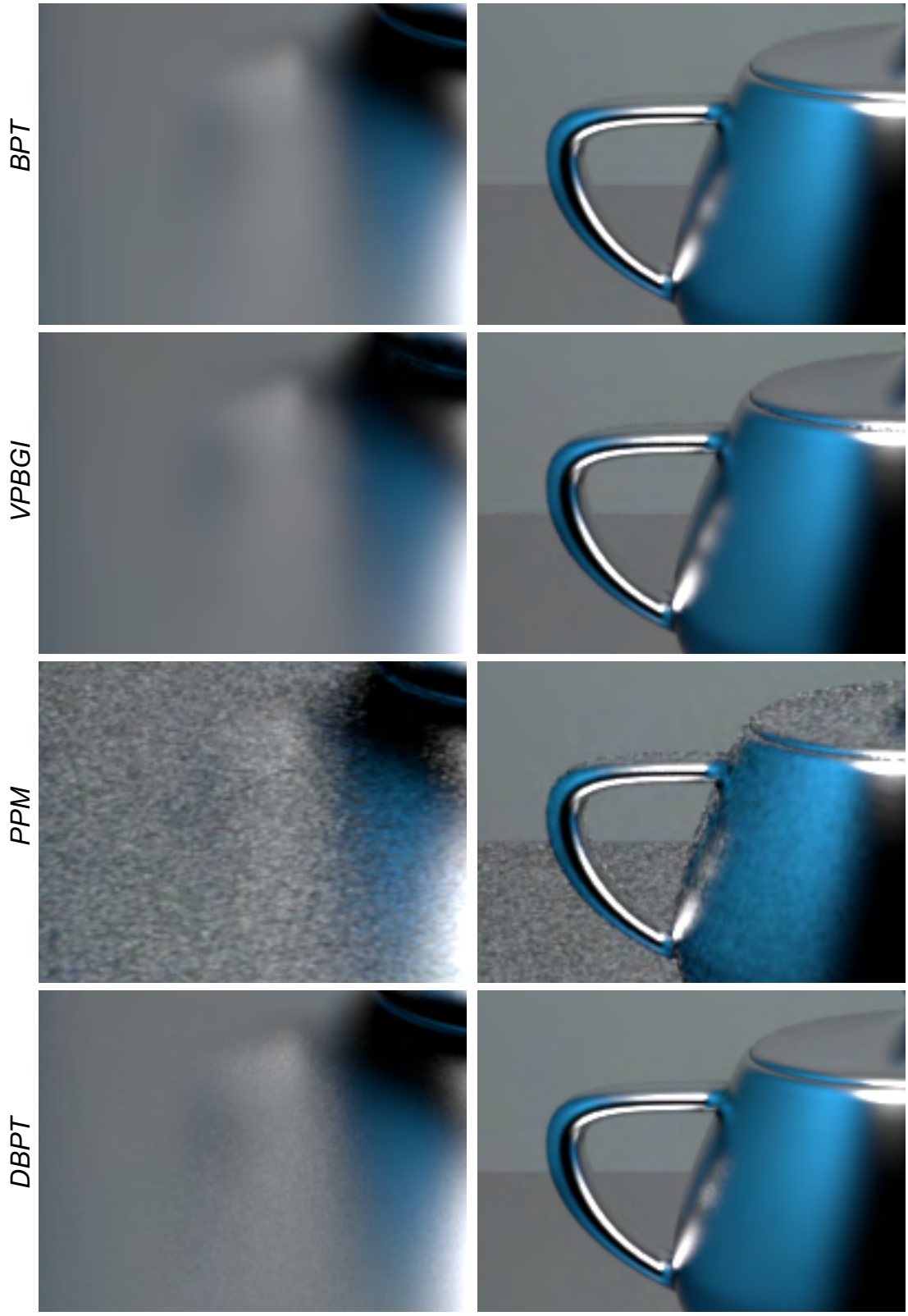


图 5.6 反射次数为 2 情况下的 Teapot 场景细节对照

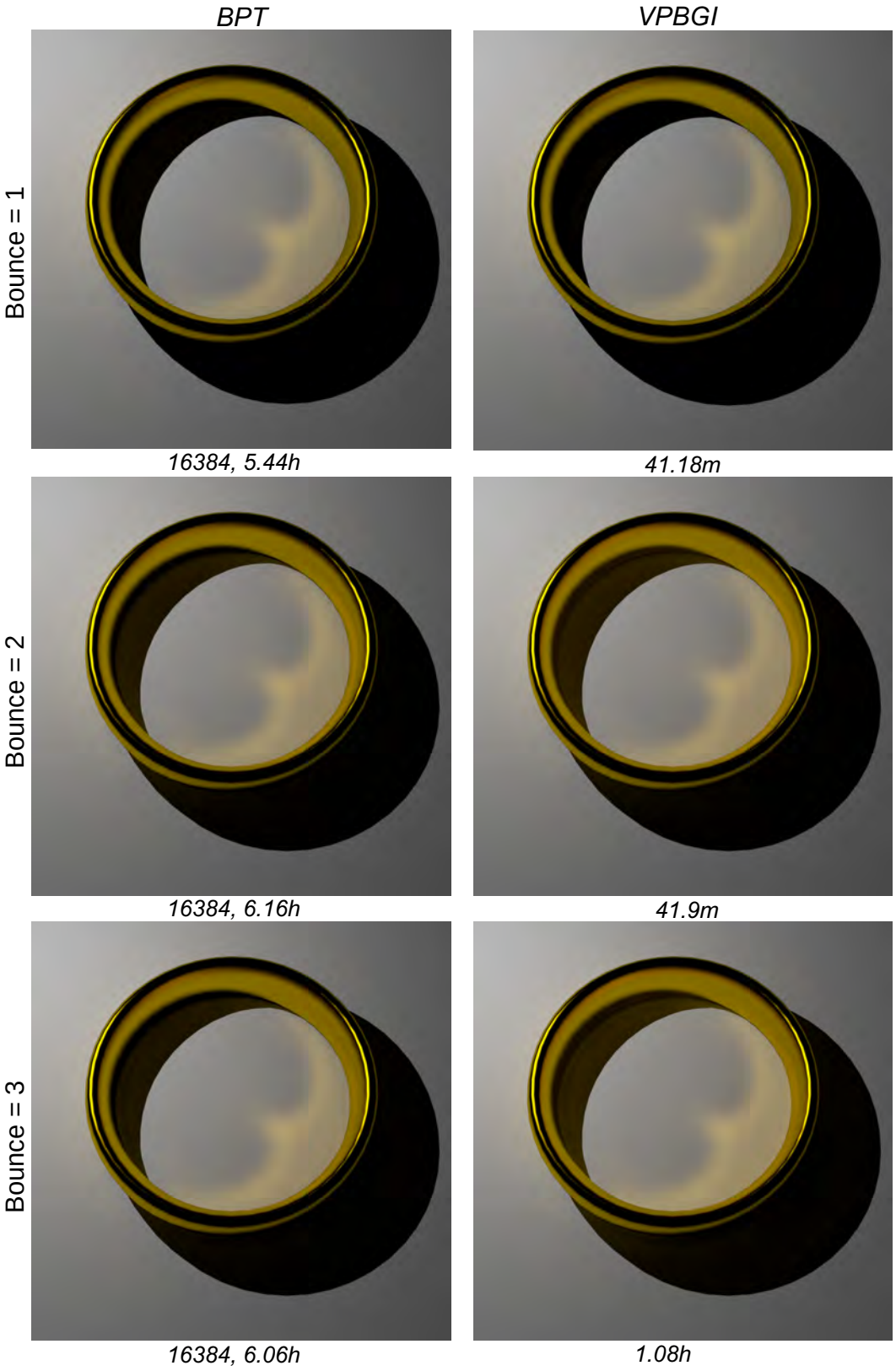


图 5.7 Ring 场景对照

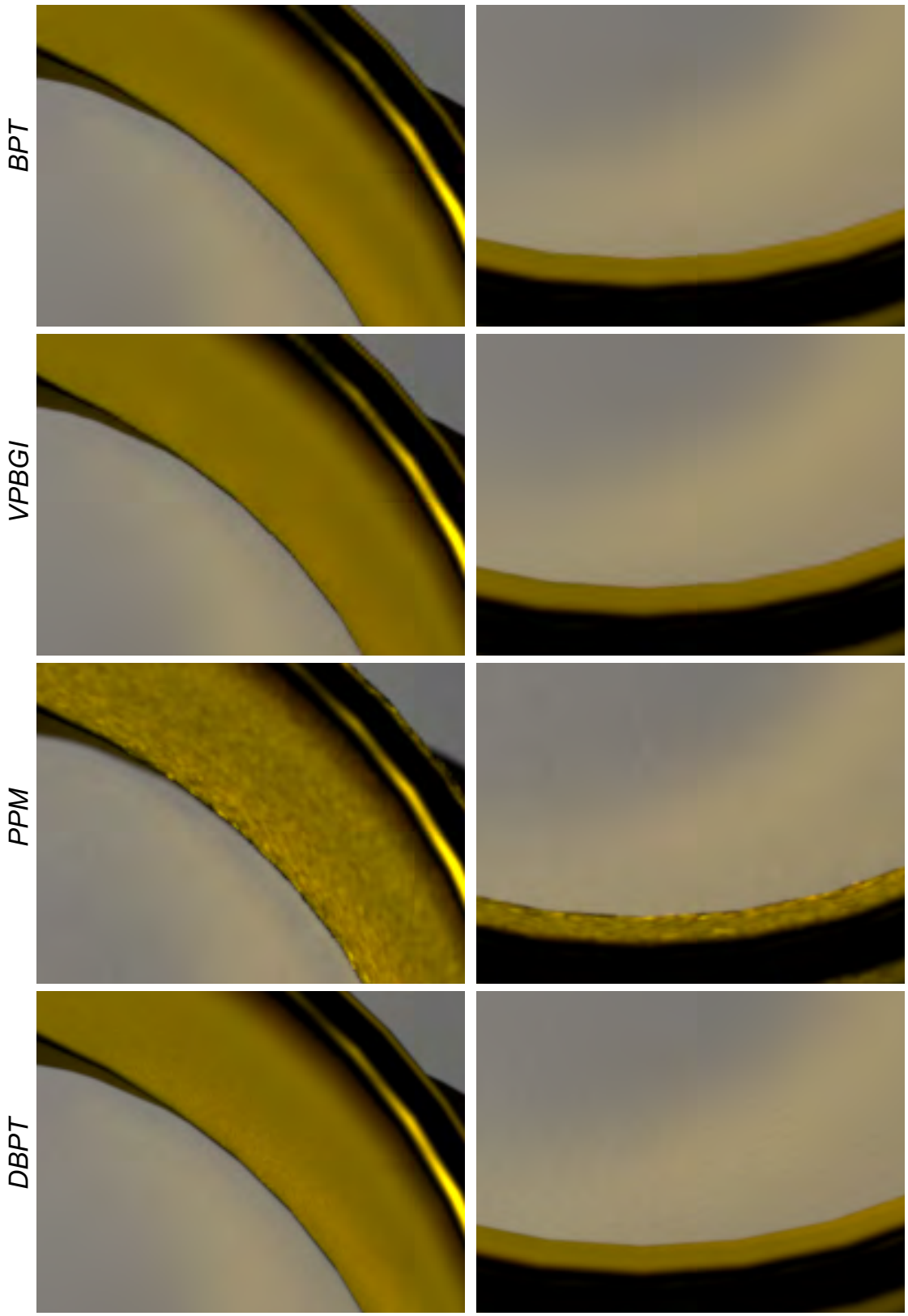


图 5.8 反射次数为 2 情况下的 Ring 场景细节对照

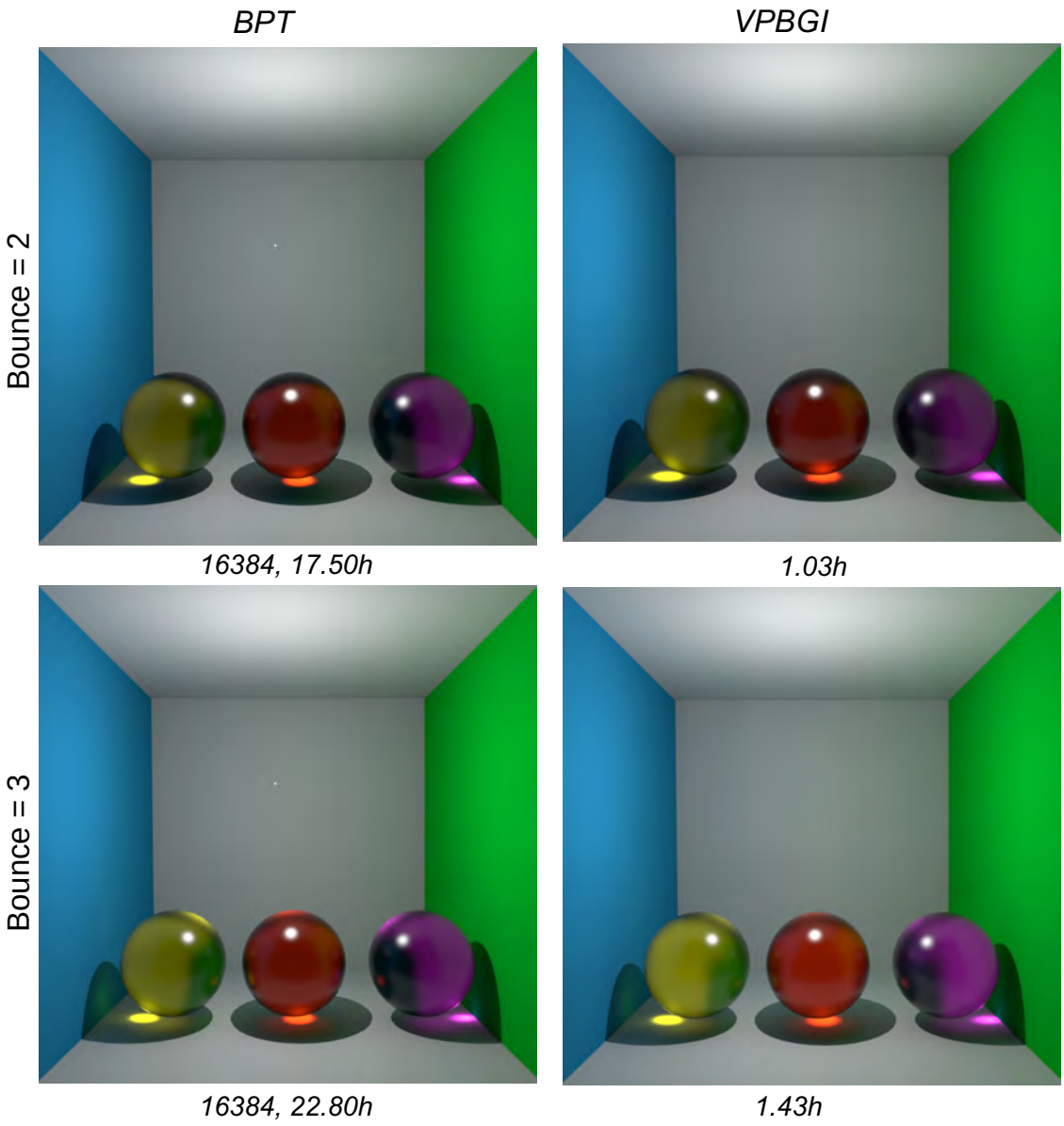


图 **5.9** Sphere 场景对照

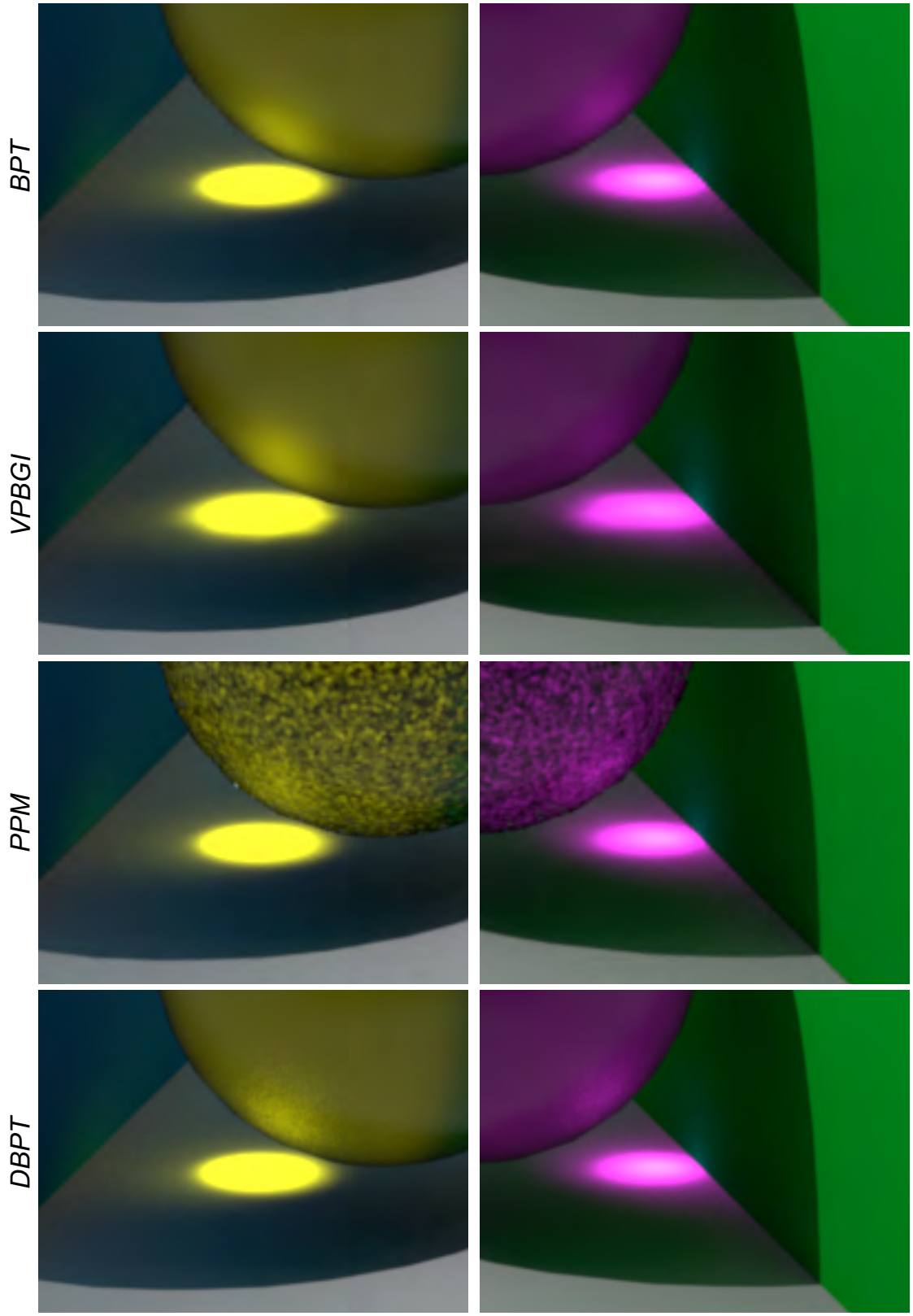
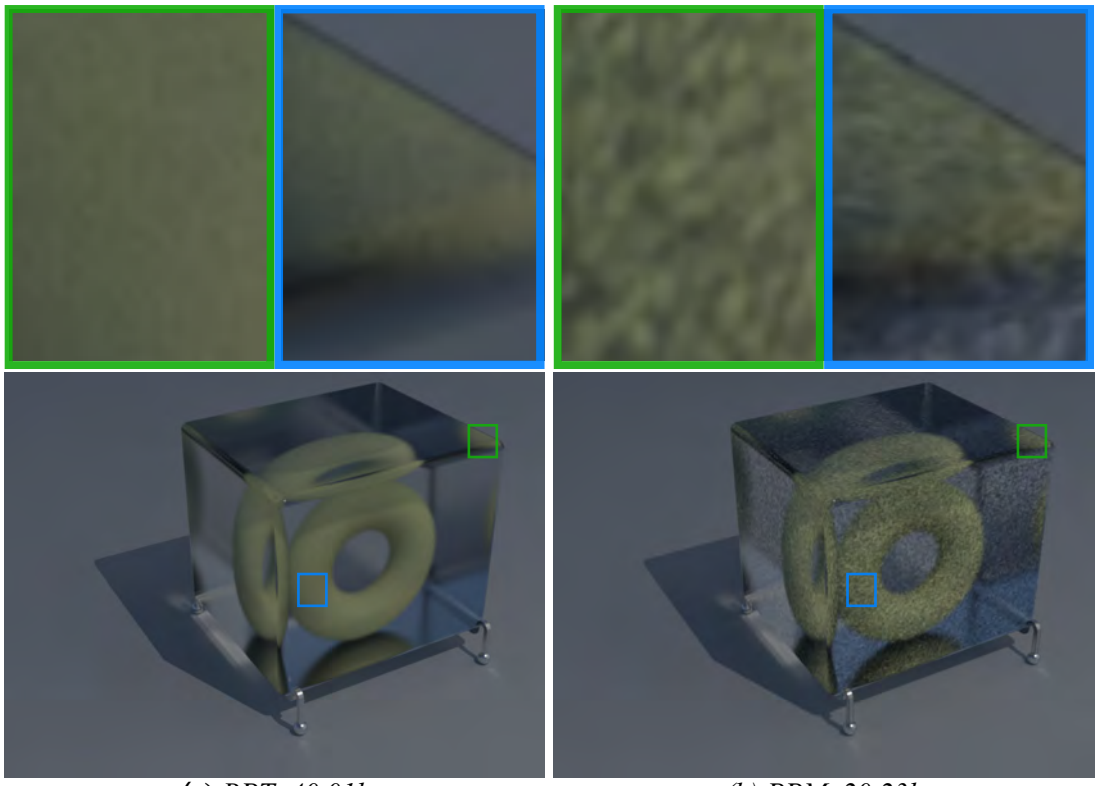
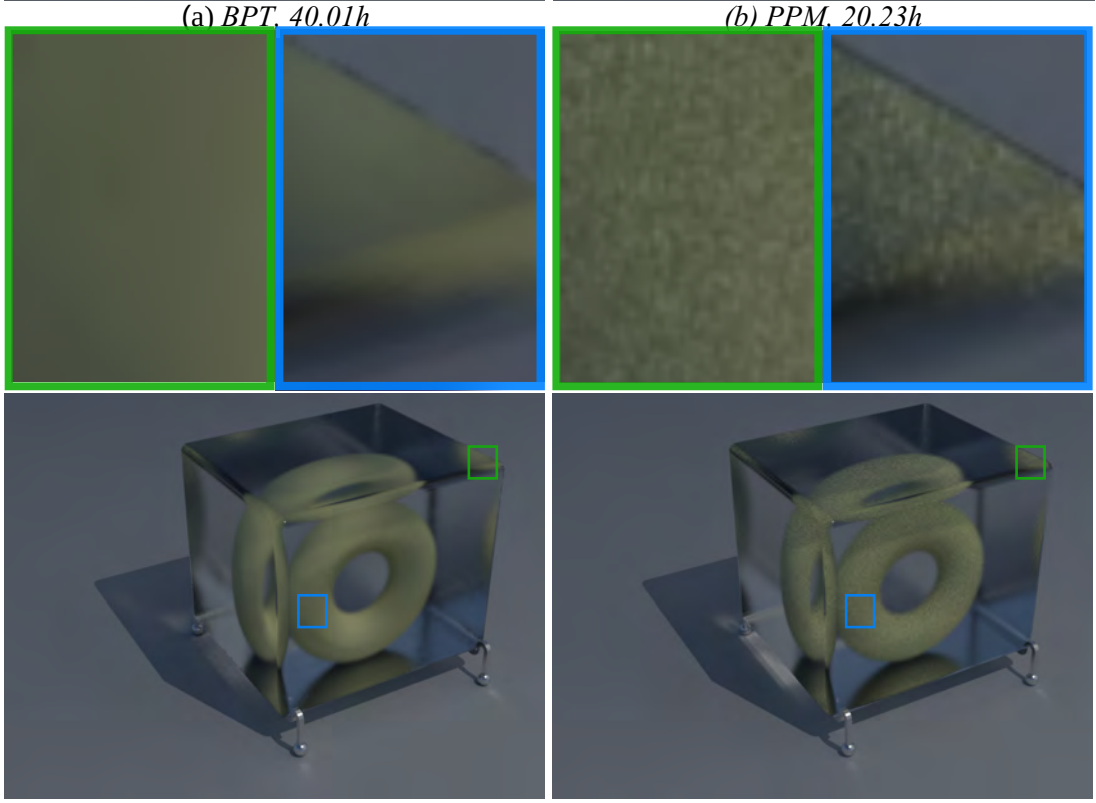


图 5.10 反射次数为 2 情况下 Sphere 场景对照



(a) *BPT, 40.01h*　　*(b) PPM, 20.23h*



(c) WPBGI, 4.37h　　*(d) DBPT, 5.17h*

图 5.11 反射次数为 3 情况下 Sphere 场景对照

性能统计 表 3 统计了测试场景在反射次数为 2 (即最大路径为 4)的情况下的渲染时间和误差情况。 其中 *pts* 表示 VPBGI 算法中点云中点的个数； *num.* 表示 BPT 算法中的采样数； *mem.* 表示在渲染过程中使用内存的峰值； *pr.* 表示点云产生和点云层次结构构建以及点云树更新的时间； *ren.* 表示着色点的渲染时间， 该时间包含了直接光照计算时间 (很短，可忽略) 和间接光照时间， 其中后者包括了微缓冲区初始化，点云层次结构遍历，点云层次结构节点投影以及微缓冲区与 BRDF 进行卷积所需要的时间； *tot.* 是以上两部分时间之和； 误差表示了 VPBGI 算法渲染结果与 BPT 算法渲染结果的均方差 (MSE)。

我们在表 5.2 把使用视点树前后的预处理时间进行对比和分析， *bounces* 表示反射的次数，bounces 为 1 表示只有第一次间接光照； *tree constr.* 表示点云树构建的时间； *travers.* 表示在该次反射中进行点云树遍历和投影的时间； *rad.* 表示出射辐射度计算时间； *tot.* 是以上两部分时间之和； 通过将基于视点树和没有使用视点树的方法进行对照，可以得出如下结论： 基于视点树的方法能够有效提高多次反射中点云树遍历和投影的时间，加速比大约为是 1.7x 到 3.5x， 在实验中，我们使用的视点立体角为 0.005。

预览渲染结果 在图 5.12 中，我们给出场景 Teapot 用点云树来进行预览渲染的结果，并且给出了不同视点，将 BPT 离线渲染的结果作为对照。 通过对比，我们得出结论利用点云树来进行预览渲染，结果与对照结果相似 (视觉上)，但是也有一些问题，由于直接使用出射辐射亮度，因此在插值产生着色点的出射辐射亮度时，在部分地方有走样。

5.6 总结

首先讨论我们的算法的不足之处。 第一，由于在算法中采用全局坐标系下的 6 个面的哈尔小波来表示出射辐射亮度，而在微缓冲区中采用局部坐标系下的小波系数， 由于哈尔小波不容易进行旋转，因此无法直接得到小波系数，而需要在空间域与频域之间转换， 因此采用支持旋转的球形小波 [66] 将是更好的办法。 第二，把存储了全局光照的点云层次结构应用到预览渲染中时，由于受到内存的限制， 只能将某个层次上的小波系数传输到 GPU，当将视点集中到某个细节时，可

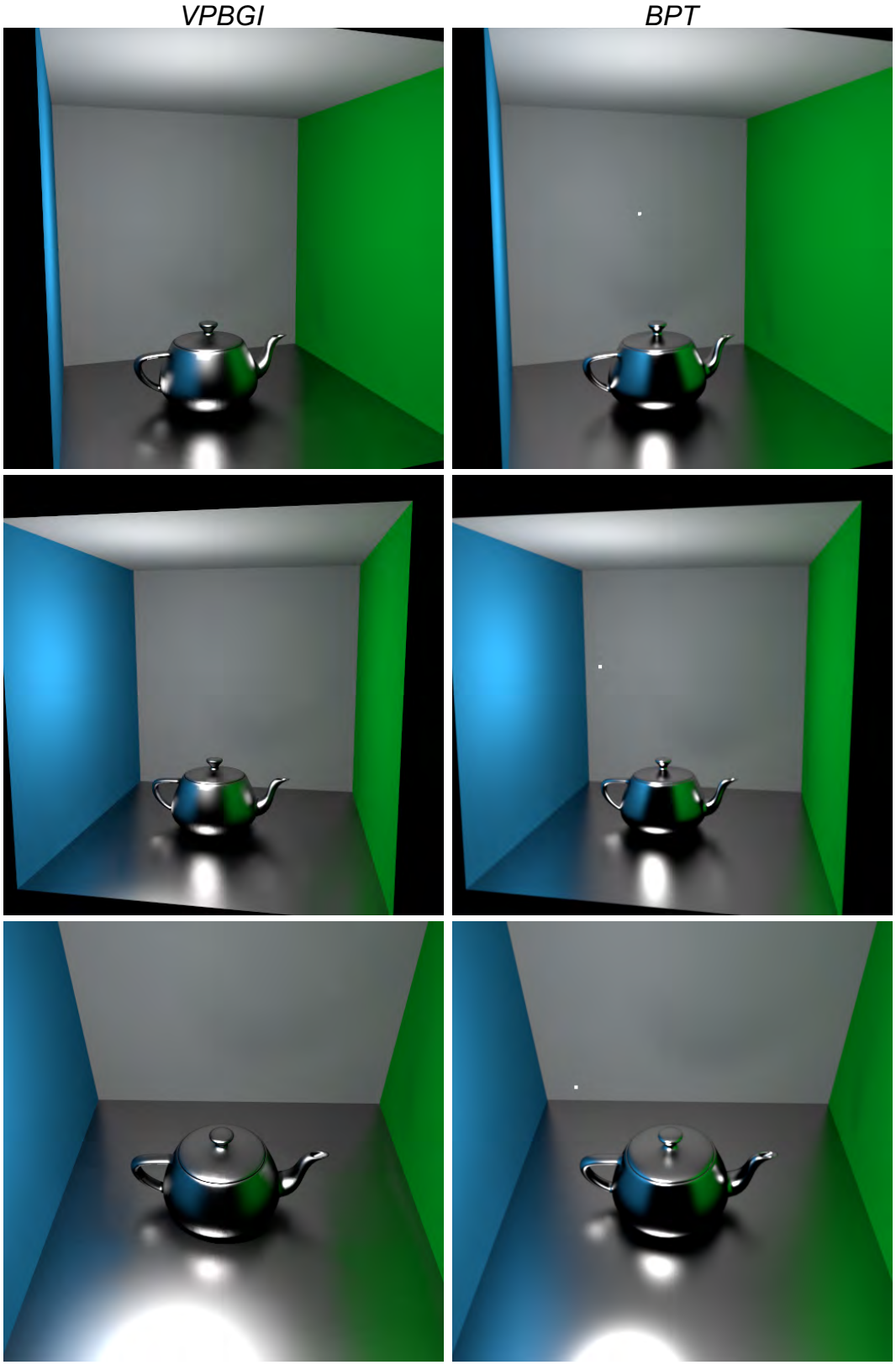


图 5.12 Teapot 在不同视点下的预览渲染

能会由于采样不足而产生走样。 第三，随着反射层数的增多，所增加的某次反弹的辐射亮度对于最终图像的贡献减低， 因此在计算出射辐射亮度时所需要的树切会更加粗糙， 而在本章中并未对该情况进行处理。

在本章中，我们提出了基于视点树的多次反射 PBGI，充分利用空间连贯性，此外，我们提出了出射辐射亮度的快速计算模型，从而进一步提高效率。 我们的算法相对于 BPT 算法，有 9 到 17 倍的提速，易于在当前的 PBGI 框架中实现，并且只有较少的可控参数。 最后，我们将提出的多次反射计算结果应用于预览渲染，在场景中支持漫反射材质以及各种频率的光泽反射材质， 另外支持视点可变。

关于本文的未来工作，有以下几点。 首先，支持层次化的预览预览，即在预览改变视点时，获取适当的树切，解决预览细节时的走样问题； 其次，目前预览渲染中只支持视点的变化，而支持光源的变化以及者材质的变化将使其具有更多的应用。

第 六 章　总结与展望

6.1 总结

真实感图形渲染的目标是将现实中的光线传递在虚拟的模型中实现，全局光照的过程正是计算光线传递的过程，因此在真实感渲染中起到至关重要的作用。由于全局光照计算的复杂性，因此需要很大的时间成本，虽然计算机硬件水平不断提高，但是效率的提高仍然是全局光照面临的重要问题。 PBGI 算法提出之后，之所以得到广泛应用，最重要的原因在于比光线跟踪算法具有更高效率。 PBGI 算法效率的进一步提高也具有重要的意义，将进一步节省资源。 但是 PBGI 存在无法模拟非漫反射光线传递的缺点，从而限制了该技术的使用范围。

在该论文中，围绕 PBGI 中存在的问题，提出了解决方案。 在第三章中提出了基于分解的方案来提高 PBGI 效率； 在第四章中对 PBGI 算法进行了扩展，使其可以模拟非漫反射光线传递； 第五章，提出基于视点树的 PBGI 多次反射计算算法，并且可以应用到视点可变的预览渲染中。 我们将对每章分别进行总结。

6.1.1 基于分解的点缓存全局光照算法

PBGI 的两个核心过程是点云层次结构遍历找到树切和节点向微缓冲区进行投影。 根据空间连贯性，相邻着色点的这两个过程存在很大的相似度。 在充分考虑到空间连贯性的基础上，我们提出了重用树切和微缓冲的技术 (FPBGI)。 首先将着色点根据位置和法向的相似程度进行聚类，从而找到具有空间连贯性的点，根据聚类中的活跃着色点找到聚类树切之后，将聚类树切中的节点分为远节点和近节点： 远节点被聚类中的所有着色点所共享，并且向聚类微缓冲区直接投影；聚类中每个着色点分别对近节点进行遍历，并且把新产生的节点或者近节点向各自的微缓冲区进行投影。 最终，对每个着色点将两个微缓冲区融合成一个微缓冲区，并且与 BRDF 卷积后得到最终的出射辐射亮度。

该技术通过对树切和微缓冲区的重用，提高了渲染效率。 在我们测试的例子中，与 PBGI 算法相比可以达到 2x 到 4x 的加速比。 另外，FPBGI 保持了 PBGI 算

法无噪声的优势，具有时间连贯性， 易于在当前的 PGBI 框架中实现，并且只使用了少量的可控的参数。

6.1.2 基于小波的点缓存全局光照算法

针对 PGBI 算法无法计算非漫反射光线传递的问题，本论文中提出了基于小波的方案 (WPBGI)。 对每个点提出基于小波的辐射亮度模型，用于表示非漫反射点的出射辐射亮度。 利用后序遍历的方法计算点云层次结构中每个节点的小波系数，从而减少内存使用。 此外，将小波系数在层次结构上进行编码或者小波分析，在层数低节点中存储小波近似系数，而在层数高节点中存储小波细节系数， 在这种编码方式基础上，再使用非线性近似，从而进一步降低内存压力。 在投影阶段，提出重要性驱动微缓冲区技术，根据光照和 BRDF 重要性函数来对微缓冲区的特定像素进行细分， 从而解决了在高频光照或者 BRDF 情况下出现的走样问题。

在我们的测试场景中，在保证相似质量的前提下， WPBGI 效率是 BPT 算法效率的 3x 到 10x。

6.1.3 PBGI 的多次反射快速计算方法

将 PBGI 应用到光照多次反射计算时，需要对点云中的每个点进行点云树的遍历和投影，本论文中提出了基于视点树的解决方案，从而达到层次化地重用点之间树切的目的，从而提高效率。 此外，在计算出射辐射亮度时，提出四维 BRDF 系数与二维入射辐射亮度系数相乘的方法， 提高了数据的稀疏性，减少了内存占用，同时提高了计算效率。 此外，将球形微缓冲区代替半球微缓冲区，从而支持折射效果的模拟。通过测试，我们的算法得到的效率是 BPT 算法效率的 9x 到 17x，没有明显的走样，误差也是可接受的。 最后，将包含了多次反射的点云树用于场景的预览渲染， 支持视点的变化，支持漫反射以及各个频率的光泽反射。

6.2 展望

在以后的工作中，我们将关于 PGBI 继续以下几个方面的研究。

内存策略 内存问题是 PBGI 的一个重要问题。目前，已有多种策略用于解决该问题， 比如 [31] 中提出的基于外存的方法， [33] 中提出的基于压缩的方法等，

本文也使用了一些策略来减少内存压力，但是在大场景或者场景中出现高频的BRDF时，仍然存在内存不够问题。关于内存问题的一种解决思路是按需产生点云，即在不需要大量细节的地方只产生少量的点。

基于PBGI的重光照技术 在本文的第五章，我们将点云树用于视点可变的预览渲染。但是在实际应用中，可变视点没有完全满足用户的要求，即灵活性不够。将PBGI用于完全可变(视点可变、材质可变、光照可变、物体可变、全频光照等)中将会有更大的应用价值。第五章中的预览渲染需要大量的时间进行预计算，来获得光照在场景中分布，由于点云是视点无关的，支持视点可变，但是当除视点之外的其他因素改变时，将需要重新进行预计算，因此无法达到实时或者可交互的效率，如何解决这个问题，将是该研究方向的核心。

参考文献

[1] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. « Radiometry ». In: 1992. Chap. Geometrical Considerations and Nomenclature for Reflectance, pp. 94–145.

[2] W. Jakob. *Mitsuba Renderer*. http://www.mitsuba-renderer.org/. 2010.

[3] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. « Microfacet Models for Refraction Through Rough Surfaces ». In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. 2007, pp. 195–206.

[4] J. T. Kajiya. « The Rendering Equation ». In: *SIGGRAPH Comput. Graph.* 20.4 (1986), pp. 143–150.

[5] P. S. Heckbert. « Adaptive Radiosity Textures for Bidirectional Ray Tracing ». In: *SIGGRAPH Comput. Graph.* 24.4 (1990), pp. 145–154.

[6] B. BUCHHOLZ. « Abstraction et Traitement de Masses de Données 3D Animées ». PhD thesis. TELECOM ParisTech, 2012.

[7] T. Whitted. « An improved illumination model for shaded display ». In: *Communications of the ACM* 23.6 (1980), pp. 343–349.

[8] E. P. Lafortune and Y. D. Willems. « Bi-Directional Path Tracing ». In: *COMPUGRAPHICS ' 93*. 1993, pp. 145–153.

[9] E. Veach and L. Guibas. « Bidirectional Estimators for Light Transport ». In: 1994, pp. 147–162.

[10] E. Veach and L. J. Guibas. « Metropolis Light Transport ». In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. 1997, pp. 65–76.

[11] J. Lehtinen, T. Karras, S. Laine, M. Aittala, F. Durand, and T. Aila. « Gradient-domain Metropolis Light Transport ». In: *ACM Trans. Graph.* 32.4 (July 2013), 95:1–95:12.

[12] J. Vorba, O. Karlík, M. Šik, T. Ritschel, and J. Křivánek. « On-line Learning of Parametric Mixture Models for Light Transport Simulation ». In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)* 33.4 (2014).

[13] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. « Modeling the Interaction of Light Between Diffuse Surfaces ». In: *SIGGRAPH Comput. Graph.* 18.3 (1984).

[14] D. S. Immel, M. F. Cohen, and D. P. Greenberg. « A Radiosity Method for Non-diffuse Environments ». In: *SIGGRAPH Comput. Graph.* 20.4 (1986), pp. 133–142.

[15] J. R. Wallace, M. F. Cohen, and D. P. Greenberg. « A Two-pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods ». In: *SIGGRAPH Comput. Graph.* 21.4 (1987), pp. 311–320.

[16] P. Hanrahan, D. Salzman, and L. Aupperle. « A Rapid Hierarchical Radiosity Algorithm ». In: *SIGGRAPH Comput. Graph.* 25.4 (1991), pp. 197–206.

[17] H. W. Jensen. « Global Illumination Using Photon Maps ». In: *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. 1996, pp. 21–30.

[18] T. Hachisuka, S. Ogaki, and H. W. Jensen. « Progressive Photon Mapping ». In: *ACM Trans. Graph.* 27.5 (2008), 130:1–130:8.

[19] T. Hachisuka and H. W. Jensen. « Robust adaptive photon tracing using photon path visibility ». In: *ACM Trans. Graph.* (2011), pp. 114–114.

[20] A. Keller. « Instant radiosity ». In: *ACM SIGGRAPH 1997*. 1997, pp. 49–56.

[21] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg. « Lightcuts: A Scalable Approach to Illumination ». In: *ACM Trans. Graph.* 24.3 (2005), pp. 1098–1107.

[22] M. Hašan, J. Křivánek, B. Walter, and K. Bala. « Virtual Spherical Lights for Many-light Rendering of Glossy Scenes ». In: *ACM Trans. Graph.* 28.5 (2009), 143:1–143:6.

[23] B. Walter, P. Khungurn, and K. Bala. « Bidirectional Lightcuts ». In: *ACM Trans. Graph.* 31.4 (2012), 59:1–59:11.

[24] P Christensen. *Point-based approximate color bleeding*. Tech. rep. 08-01. Pixar Technical Notes, 2008.

[25] J. Křivánek, M. Fajardo, P. H. Christensen, E. Tabellion, M. Bunnell, D. Larsson, and A. Kaplanyan. « Global Illumination Across Industries ». In: *ACM SIGGRAPH 2010 Courses*. SIGGRAPH '10. 2010.

[26] C. R. Vogel. *Computational Methods for Inverse Problems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002.

[27] L.-Y. Wei. « Parallel Poisson Disk Sampling ». In: *ACM Trans. Graph.* 27.3 (Aug. 2008), 20:1–20:9.

[28] D. Dunbar and G. Humphreys. « A Spatial Data Structure for Fast Poisson-disk Sample Generation ». In: *ACM Trans. Graph.* 25.3 (2006), pp. 503–508.

[29] J. Chen, X. Ge, L.-Y. Wei, B. Wang, Y. Wang, H. Wang, Y. Fei, K.-L. Qian, J.-H. Yong, and W. Wang. « Bilateral Blue Noise Sampling ». In: *ACM Trans. Graph.* 32.6 (2013), 216:1–216:11.

[30] R. L. Cook, L. Carpenter, and E. Catmull. « The Reyes Image Rendering Architecture ». In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 95–102.

[31] J. Kontkanen, E. Tabellion, and R. S. Overbeck. « Coherent out-of-core point-based global illumination ». In: *Comp. Graph. Forum (Proc. EGSR 2011)*. 2011, pp. 1353–1360.

[32] T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, and C. Dachsbacher. « Micro-Rendering for Scalable, Parallel Final Gathering ». In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia 2009)* 28.5 (2009).

[33] B. Buchholz and T. Boubekeur. « Quantized Point-Based Global Illumination ». In: *Comp. Graph. Forum (Proc. EGSR 2012)* 31.4 (2012), pp. 1399–1405.

[34] B. Wang, J. Huang, B. Buchholz, X. Meng, and T. Boubekeur. « Factorized Point-Based Global Illumination ». In: *Computer Graphics Forum (Special Issue on EGSR 2013)* 32.4 (2013), pp. 117–123.

[35] D. Maletz and R. Wang. « Importance Point Projection for GPU-based Final Gathering ». In: *Comp. Graph. Forum* 30.4 (2011), pp. 1327–1336.

[36] M. Holländer, T. Ritschel, E. Eisemann, and T. Boubekeur. « ManyLoDs: Parallel Many-View Level-of-Detail Selection for Real-Time Global Illumination ». In: *Comp. Graph. Forum (Proc. EGSR 2011)* 30.4 (2011), pp. 1233–1240.

[37] E. Tabellion. « Point-Based Global Illumination Directional Importance Mapping ». In: *ACM SIGGRAPH Talk*. 2012.

[38] G. J. Ward, F. M. Rubinstein, and R. D. Clear. « A ray tracing solution for diffuse interreflection ». In: *ACM SIGGRAPH Computer Graphics*. Vol. 22. 4. 1988, pp. 85–92.

[39] G. J. Ward and P. Heckbert. « Irradiance gradients ». In: *Eurographics Workshop on Rendering*. 1992.

[40] R. Wang, R. Wang, K. Zhou, M. Pan, and H. Bao. « An efficient GPU-based approach for interactive global illumination ». In: *ACM Trans. Graph. (Proc. SIGGRAPH 2009)*. 2009, 91:1–91:8.

[41] B. Wang, X. Meng, Y. Xu, and X. Song. « Fast Point Based Global Illumination ». In: *Computer-Aided Design and Computer Graphics*. 2011, pp. 93–98.

[42] J. Krivanek, P. Gautron, S. Pattanaik, and K. Bouatouch. « Radiance caching for efficient global illumination computation ». In: *IEEE TVCG* 11.5 (2005), pp. 550–561.

[43] P. Shanmugam and O. Arikan. « Hardware accelerated ambient occlusion techniques on GPUs ». In: *ACM I3D*. 2007, pp. 73–80.

[44] O. Arikan, D. A. Forsyth, and J. F. O'Brien. « Fast and detailed approximate global illumination by irradiance decomposition ». In: *ACM SIGGRAPH 2005*. 2005, pp. 1108–1114.

[45] D. Cohen-Steiner, P. Alliez, and M. Desbrun. « Variational shape approximation ». In: *ACM Trans. Graph.* 23.3 (2004), pp. 905–914.

[46] H. Yee. « A perceptual metric for production testing ». In: *Journal of Graphics Tools* 9.4 (2004), pp. 33–40.

[47] E. J. Stollnitz, T. D. Derose, and D. H. Salesin. « Wavelets for Computer Graphics: A Primer - Part 1 ». In: *IEEE Computer Graphics and Applications* 15 (1995), pp. 76–84.

[48] R. Green. « Spherical harmonic lighting: The gritty details ». In: *Archives of the Game Developers Conference*. Vol. 5. 2003.

[49] S. J. Gortler, P. Schröder, M. F. Cohen, and P. Hanrahan. « Wavelet Radiosity ». In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. 1993, pp. 221–230.

[50] P. Christensen, E. Stollnitz, D. Salesin, and T. DeRose. « Wavelet radiance ». In: *Photorealistic Rendering Techniques*. Springer, 1995, pp. 295–309.

[51] P.-P. Sloan, J. Kautz, and J. Snyder. « Precomputed Radiance Transfer for Real-time Rendering in Dynamic, Low-frequency Lighting Environments ». In: *ACM Trans. Graph.* 21.3 (2002), pp. 527–536.

[52] R. Ng, R. Ramamoorthi, and P. Hanrahan. « All-frequency Shadows Using Non-linear Wavelet Lighting Approximation ». In: *ACM Trans. Graph.* 22.3 (2003), pp. 376–381.

[53] R. Ng, R. Ramamoorthi, and P. Hanrahan. « Triple product wavelet integrals for all-frequency relighting ». In: *ACM Trans. Graph.* 23.3 (2004), pp. 477–487.

[54] W. Sun and A. Mukherjee. « Generalized Wavelet Product Integral for Rendering Dynamic Glossy Objects ». In: *ACM Trans. Graph.* 25.3 (2006), pp. 955–966.

[55] J. Kontkanen, E. Turquin, N. Holzschuch, and F. Sillion. « Wavelet Radiance Transport for Interactive Indirect Lighting ». In: *Rendering Techniques 2006 (Eurographics Symposium on Rendering)*. 2006.

[56] M. Hašan, F. Pellacini, and K. Bala. « Direct-to-indirect Transfer for Cinematic Relighting ». In: *ACM Trans. Graph.* 25.3 (2006), pp. 1089–1097.

[57] P. Lalonde. « A Wavelet Representation of the BRDF ». In: *IEEE Transactions on Visualization and Computer Graphics* 3 (1997), pp. 329–336.

[58] L. Claustres, M. Paulin, and Y. Boucher. « BRDF Measurement Modelling using Wavelets for Efficient Path Tracing ». In: *Computer Graphics Forum* 22 (2003), pp. 701–716.

[59] R. Ramamoorthi and P. Hanrahan. « An Efficient Representation for Irradiance Environment Maps ». In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. 2001, pp. 497–500.

[60] P. Green, J. Kautz, W. Matusik, and F. Durand. « View-dependent precomputed light transport using nonlinear Gaussian function approximations ». In: *SI3D '06*. 2006, pp. 7–14.

[61] P. Green, J. Kautz, and F. Durand. « Efficient Reflectance and Visibility Approximations for Environment Map Rendering ». In: *Computer Graphics Forum (Proc. EUROGRAPHICS)* 26.3 (2007), pp. 495–502.

[62] J. Wang, P. Ren, M. Gong, J. Snyder, and B. Guo. « All-frequency Rendering of Dynamic, Spatially-varying Reflectance ». In: *ACM Trans. Graph.* 28.5 (Dec. 2009), 133:1–133:10.

[63] K. Xu, W.-L. Sun, Z. Dong, D.-Y. Zhao, R.-D. Wu, and S.-M. Hu. « Anisotropic Spherical Gaussians ». In: *ACM Transactions on Graphics* 32.6 (2013), 209:1–209:11.

[64] Y.-T. Tsai and Z.-C. Shih. « All-frequency Precomputed Radiance Transfer Using Spherical Radial Basis Functions and Clustered Tensor Approximation ». In: *ACM Trans. Graph.* 25.3 (2006), pp. 967–976.

[65] J. Shapiro. « Embedded Image Coding Using Zerotrees of Wavelet Coefficients ». In: *Trans. Sig. Proc.* 41.12 (1993), pp. 3445–3462.

[66] P. Schröder and W. Sweldens. « Spherical Wavelets: Efficiently Representing Functions on the Sphere ». In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. ACM, 1995, pp. 161–172.

[67] T. Saito and T. Takahashi. « Comprehensible Rendering of 3-D Shapes ». In: *SIGGRAPH Comput. Graph.* 24.4 (Sept. 1990), pp. 197–206.

[68] C. Dachsbacher and M. Stamminger. « Reflective Shadow Maps ». In: *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. I3D '05. Washington, District of Columbia, 2005, pp. 203–231.

[69] C. Dachsbacher and M. Stamminger. « Splatting Indirect Illumination ». In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. I3D '06. Redwood City, California, 2006, pp. 93–100.

[70] G. Nichols and C. Wyman. « Multiresolution Splatting for Indirect Illumination ». In: *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. I3D '09. Boston, Massachusetts, 2009, pp. 83–90.

[71] G. Nichols and C. Wyman. « Interactive Indirect Illumination Using Adaptive Multiresolution Splatting ». In: *IEEE Transactions on Visualization and Computer Graphics* 16.5 (2010), pp. 729–741.

[72] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann. « Interactive Indirect Illumination Using Voxel Cone Tracing ». In: *Computer Graphics Forum (Proc. of Pacific Graphics)* 30.7 (2011).

[73] P. Gautron, J. Křivánek, K. Bouatouch, and S. N. Pattanaik. « Radiance Cache Splatting: A GPU-Friendly Global Illumination Algorithm ». In: *Rendering Techniques 2005, Eurographics Symposium on Rendering, Proceedings*. 2005, pp. 55–64.

[74] N. Greene. « Environment Mapping and Other Applications of World Projections ». In: *IEEE Comput. Graph. Appl.* 6.11 (Nov. 1986), pp. 21–29.

[75] W. Heidrich and H.-P. Seidel. « Realistic, Hardware-accelerated Shading and Lighting ». In: *Computer Graphics (SIGGRAPH-99) : Conference proceedings*. Ed. by A. Rockwood. ACM Siggraph. Los Angeles, USA: ACM, 1999, pp. 171–178.

[76] D. Scherzer, C. H. Nguyen, T. Ritschel, and H.-P. Seidel. « Pre-convolved Radiance Caching ». In: *Comp. Graph. Forum* 31.4 (June 2012), pp. 1391–1397.

[77] M. Bunnell. « Dynamic ambient occlusion and indirect lighting ». In: *GPU Gems* 2 (2005), pp. 223–233.

[78] J. Laurijssen, R. Wang, P. Dutré, and B. J. Brown. « Fast Estimation and Rendering of Indirect Highlights ». In: *Proceedings of the 21st Eurographics Conference on Rendering*. 2010, pp. 1305–1313.

[79] J. Lehtinen, M. Zwicker, E. Turquin, J. Kontkanen, F. Durand, F. X. Sillion, and T. Aila. « A Meshless Hierarchical Representation for Light Transport ». In: *ACM Trans. Graph.* 27.3 (2008), 37:1–37:9.

致谢

十年弹指一挥间，在山东大学即将满十个年头。 在硕博连读的这五年半时间中，有太多收获，也有太多感谢。 首先，我们要感谢我的导师孟祥旭教授对我的谆谆教导，不仅指导我们如何做好学术，更加是教导我们如何在科研的道路上走得更远； 感谢徐延宁老师给予我的指导； 感谢我在 Telecom ParisTech 的导师 Tamy Boubekeur 教授给予我的两年指导； 感谢渲染组的各位老师和同学们以及山东大学人机交互与虚拟现实实验室的老师们给予的帮助和支持； 感谢 Telecom 的朋友和同事们给予的帮助； 最后，我也特别感谢，我的家人们给予的支持和开导；感谢我的朋友们给予我的生活上照顾和学术上的支持，特别感谢江城和孙强。

攻读博士期间主要的研究成果

发表论文

1. Beibei Wang, Yanning Xu, and Xiangxu Meng. Progressive Point Based Global Illumination. *Workshop on Digital Media and Digital Content Management 2011*, pp.181-184. (EI检索号：20113414253308),第一位。
2. Beibei Wang, Xiangxu Meng, Yanning Xu, and Xijun Song. Fast Point Based Global Illumination. *Computer-Aided Design and Computer Graphics 2011*, pp.93-98. (EI检索号：20114714531290). 第一位。
3. Beibei Wang, Zhen Xu, Yanning Xu, and Xiangxu Meng. Efficient Point Based Global Illumination on GPU. *Eurographics 2012 – Posters*, pp.17-18. 第一位。
4. Beibei Wang, Jing Huang, Bert Buchholz, Xiangxu Meng and Tamy Boubekeur. Factorized Point Based Global Illumination. *EGSR 2013 - Computer Graphics Forum Journal*, vol. 32.4(2013), pp.117-123. (SCI检索号：WOS：000321946800013)，第一位。
5. Beibei Wang, Xiangxu Meng and Tamy Boubekeur. Wavelet Point Based Global Illumination. 手稿，第一位。

参与项目

1. 真实感动漫渲染系统研究与应用，国家863项目，项目编号：2012AA01A306，参与研发。
2. 动漫创作及渲染平台研发，山东省信息产业专项发展资金项目，项目编号：2008R0047，参与研发。

外文论文 Factorized Point based Global Illumination

Abstract

The Point-Based Global Illumination (PBGI) algorithm is composed of two major steps: a caching step and a multiview rasterization step. At caching time, a dense point-sampling of the scene is shaded and organized in a spatial hierarchy, with internal nodes approximating the radiance of their subtrees using spherical harmonics. At rasterization time, a microbuffer is instantiated at the unprojected position of each image pixel (receiver). Then, a view-adaptive level-of-detail of the scene is extracted in the form of a tree cut and rasterized in the receiver's microbuffer, solving for visibility using a local variant of the z-buffer. Finally, the pixel color is computed by convolving its filled microbuffer with the surface BRDF. This noise-free indirect lighting method is widely used in the industry and captures several critical lighting effects, including ambient occlusion, color bleeding, (indirect) soft-shadows and environment lighting. However, we observe a large redundancy in this algorithm, both in cuts and receivers' microbuffers, which stems from their relatively low resolution. In this paper, we propose an evolution of PBGI which exploits spatial coherence to reduce these redundant computations. Starting from a similarity-based variational clustering of the receivers, we compute a single tree cut and rasterize a single microbuffer for each cluster. This per-cluster microbuffer provides a faithful approximation of the incident radiance for distant nodes and is composited over a receiver-specific microbuffer rasterizing only the closest nodes of the cluster's cut. This factorized approach is easy to integrate in any existing PBGI implementation and offers a significant rendering speed-up for a negligible and controllable approximation error.

1 Introduction

The visual impact of global illumination (GI) in a synthesized picture is the sum of a number of lighting effects stemming from indirect light bounces. Among them, one-bounce diffuse effects, such as ambient occlusion, directional occlusion, color bleeding and

indirect soft shadows, carry a large portion of the visual realism that typical GI solutions bring. Point-based global illumination (PBGI) is a popular rendering technique which captures such a subset of GI effects for a moderate amount of time and is intensively used in special effects and computer animation productions. This GI approximation model can be seen as a generalized forward rendering method which combines a fast adaptive approximation of the scene with a multiview rasterization. The resulting algorithm is noise-free, amenable to a parallel implementation and can even be extended to other GI effects (e.g., multiple bounces), although still away from a full GI solution, in particular when it comes to specular indirect phenomena (i.e., caustics).

1.1 Basic Algorithm

PBGI [24] runs in a two-step process: a caching step and a multiview rasterization step. At caching time, the scene is densely point-sampled (e.g., using Poisson disks), the points are shaded from the light sources – accounting for direct shadows only – and structured in a hierarchical data structure (e.g., octree, BSH). This tree is constructed bottom-up from the shaded points, with internal nodes carrying approximations of their related sub-trees (e.g., bounding sphere, normal cone, low-degrees spherical harmonics modeling the outgoing diffuse radiance).

At rasterization time, each pixel of the final picture is shaded using a so-called *microbuffer*, which is a small hemispherical RGBZ image instantiated at the unprojected position of the pixel (or *receiver*) in the scene. For each microbuffer, a specific level-of-detail (LoD) of the scene is extracted in the form of an adaptive cut in the PBGI tree. The resulting nodes are rasterized in the microbuffer using a local variant of the z-buffer algorithm to solve for visibility. The filled microbuffer is finally convolved with the point's BRDF to shade the pixel.

The two key ideas of this algorithm are (i) the point's hierarchy, which acts as an economic substitute to the actual scene when it comes to the many adaptive LoDs which have to be extracted; (ii) the microbuffers, which extend the concept of rasterization to a per-pixel/receiver level.

1.2 Redundancy Issue

Looking back at the rasterization step, we observe that a specific cut is computed from the entire scene for each single receiver. However, the resolution of their microbuffers is typically low (from 4x4 to 64x64 in practice) which immediately translates into tree cuts having a large number of coarse nodes, therefore being highly similar for nearby receivers. As we will show later, this abundant redundancy has a significant impact on the overall rendering time.

1.3 Overview

We tackle this problem by exploiting the microbuffers'spatial coherence to factorize both cut computations and rasterizations. Our factorized PBGI technique (or FPBGI) works in three steps (see Fig. 6):

1. we cluster the receivers based on their similarity and select a per-cluster *active* receiver,
2. for each cluster, we compute a single (coarser) cut from the active receiver and rasterize it in a microbuffer shared by all receivers of the cluster
3. for any receiver, we start the tree traversal from its cluster cut and rasterize only the newly added (i.e., closer) nodes in a receiver-specific microbuffer, which is composited with the cluster one before final BRDF convolution.

As a result, a large part of tree traversals and cut rasterizations are factorized among nearby receivers, which leads to an overall rendering speed-up ranging from 2x to 4x on the typical scenes illustrating this paper.

2 Previous Work

PBGI. PBGI was first introduced by Christensen [24], who proposed the idea of microbuffers and exploited the notion of point-based substitutes introduced by Bunnell [77] for real time ambient occlusion and indirect illumination. Ritschel et al. [32] then replaced cube microbuffers with 2D Lambert-warped ones, introducing importance sampling to PBGI together with an efficient GPU implementation. Holländer et al. [36] later improved

on the fine-grained parallelism of the adaptive cut computation by pairing nodes and receivers in a low-scale GPU data amplification mechanism. The cut definition itself has been addressed by Maletz and Wang [35] who used an importance-driven point projection based on an initial clustering, by Wang et al. [41] who grouped together close points with similar normals and computed average cuts for a subset of the receivers, and by Tabellion [37] who recently exposed a set of cut picking algorithms suitable for HDR imaging. The PBGI accuracy entirely depends on the density of the initial sampling and the related memory issue has been tackled by Kontkanen et al. [31], who proposed an out-of-core framework for PBGI with cache-coherent tree construction and traversal. Buchholz and Boubekeur [33] proposed an in-core solution to this problem, learning a reduced set of node data vectors in high dimension and quantizing all tree nodes against the resulting look-up table.

Coherence in rendering. Coherence through some form of "reuse" mechanism has been widely studied in GI research. Such techniques try to avoid redundancy at different levels of the GI solution computation, including irradiance, radiance, shading and even tree-cuts in a closer context to ours. Ward et al. [38] reused illumination computation by computing scalar (diffuse) irradiance on a subset of pixels and interpolating for the others, eventually using gradients [39] for smoother results. Wang et al. [40] used k-means to subsample receiving points and interpolate irradiance, reaching interactive framerates but missing small geometric details. *Radiance Caching* [42] overcomes the limitation to diffuse reflectance by storing incoming radiance as a directional function, interpolating it between pixels and convolving with the BRDF for every pixel. Closer to PBGI methods, Holländer et al. [36] proposed a **time**-coherent cut update, together with a lazy scheme bounding the amount of time dedicated to this update. Our approach is inspired by this method, but acts in the spatial domain.

Near-far decomposition. The idea of near-far irradiance decomposition has been previously studied in the context of hardware ambient occlusion [43] and final gathering [44].

Acting in a PBGI context, our approach differs in the sense that the near-far split is entirely formulated through the cluster/receiver cut, the far component being shared by numerous receivers.

3 Factorized Point Based Global Illumination

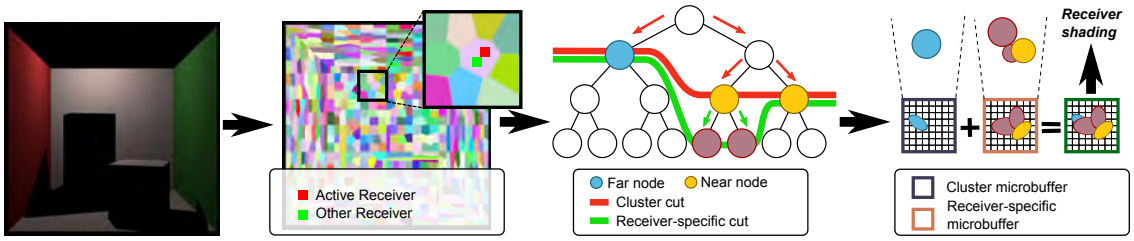


Figure 1 Principle. Starting from a tiled set of image pixels/receivers (left), we perform a variational clustering based on positional and normal similarity (middle left). For a given cluster, we compute a shared cut (middle right, red) later reused by each individual receiver to further refine their own cuts (middle right, green). The far nodes of the cluster are rasterized into a shared cluster microbuffer (right, purple) and refined nodes (added on a per-receiver basis) are rasterized in a receiver-specific microbuffer (right, orange), which is composited into one cluster for final BRDF convolution (pixel indirect shading).

3.1 Variational Receiver Clustering

Our basic assumption is that receivers with similar positions and normals have similar cuts: we propose to model this position/normal similarity by computing a variational clustering of the receivers based on a specific metric $\mathcal{D}$. To ease parallel computation, we start by regularly tiling the image space and work independently on each tile. Within a tile, we group spatially coherent receivers in k clusters using a variant of the *k-means* algorithm:

1. we initialize k centers from randomly selected receivers in the tile,
2. we cluster the tile's receivers by associating each of them to its *closest* center w.r.t. $\mathcal{D}$
3. we update clusters' centers and restart in (2).

We perform this procedure for a prescribed number of iterations and search, for each cluster, the closest receiver to the resulting center: in the following, we call it the *active* receiver of a cluster.

Following Cohen-Steiner et al. [45], we define our position/normal metric $\mathcal{D}$ as a Sobolev summed metric:

$$\mathcal{D}(\mathbf{x}, \mathbf{c}) = \|\mathbf{p_c} - \mathbf{p_x}\|^2 + \alpha\|\mathbf{n_c} - \mathbf{n_x}\|^2$$

with $\mathbf{x}$ being a receiver, $\mathbf{c}$ a cluster center, $\mathbf{p}$ (resp. $\mathbf{n}$) their position (resp. normal) in $\mathbb{R}^3$. The weight α trades cluster flatness for spatial extent. We typically set it to the length of the tile's receivers' bounding box diagonal. Last, at each iteration, the center position and normal of a cluster C are updated as follows:

$$\mathbf{p_c} = \frac{\sum_{\mathbf{x}\in C} \mathbf{p}_x}{card(C)} \quad \mathbf{n_c} = \frac{\sum_{\mathbf{x}\in C} \mathbf{n_x}}{\|\sum_{\mathbf{x}\in C} \mathbf{n_x}\|}$$

3.2 Cluster Cut and Microbuffer

Within a cluster C, the factorized workload among receivers takes the form of a single shared cut and a single microbuffer which are computed w.r.t. the *active* receiver $\mathbf{x}_C$.

In the next step of the rasterization phase, we start by traversing the PBGI tree from the root for $\mathbf{x}_C$ but stop early to produce a cut which is coarser than required in the vicinity of $\mathbf{x}_C$. Indeed, we assume that the significant difference between two nearby microbuffers only appears at fine scale (i.e., closer nodes) and deal with it later.

During the top-down PBGI tree traversal, we use a *far/near* classification of the tree's nodes based on a measure γ for each node/receiver pair: far nodes ($\gamma > \epsilon$) are traversed as usual, while near nodes ($\gamma \leq \epsilon$) stop the traversal immediatly. The node/receiver measure is defined as $\gamma = \frac{r}{d}$ with r being the cluster's radius r and d the distance between the node and $\mathbf{x}_C$. The resulting cluster cut contains two types of nodes: *far* nodes, which are rasterized in the shared cluster microbuffer, and *near* nodes, which will be concurrently refined for each individual receiver in the next step. At this stage, the cluster microbuffer already carries the distant irradiance shared by all cluster receivers.

Table 1 Performance measures.

Scene	Points	Individual Timings					Total Time		Error	
		Clustering	Cut Computation		Micro-Rasterization		*Full Rendering*			
		Time(s)	*PBGI*	*FPBGI*	*PBGI*	*FPBGI*	*PBGI*	*FPBGI*	*PDP*	*MSE*
CornellBox	88.88K	0.68s	2.46m	0.65m	1.73m	1.19m	5.72m	2.60m	103	8.79e-6
Bunny	1.00M	0.87s	2.38m	0.56m	1.55m	1.08m	4.49m	2.03m	61	1.31e-4
ItalianCity	8.38M	0.87s	3.65m	1.08m	1.23m	0.64m	5.52m	2.34m	235	8.15e-5
Sponza	16.19M	0.87s	19.71m	3.77m	5.40m	1.68m	26.72m	7.04m	15	2.51e-5

3.3 Receiver Cut, Microbuffer and Shading

In the last part of our algorithm, we process each individual receiver in parallel. For a given receiver, we compute its specific cut starting from the cluster cut (instead of the tree's root) and traversing the hierarchy down to the classical microbuffer-dependent solid angle threshold. Only the newly added nodes to the cut are marked as *refined*. Once the cut is completed, we rasterize its *refined* and *near* nodes into a receiver-specific microbuffer. Basically, only the closer nodes are rasterized and we obtain a sparse microbuffer.

Last, we composite this receiver microbuffer with the corresponding cluster one, using the depth component of both microbuffers to properly cull the microbuffer pixels which are hidden by this combination. The resulting composited microbuffer is finally convolved with the receiver's BRDF to shade the receiver/pixel.

4 Results

We implemented our technique in the Mitsuba Renderer [2], with the initial point set being generated using Poisson Disk sampling. Comparisons are performed against the original PBGI algorithm [24] and performances are measured on a 2.67GHz Intel i7 (8 cores) with 9GB of main memory. Images are rendered with one-bounce indirect illumination at a 1280×1000 pixels resolution (except for the Cornell Box, at 1024×1024) with 32×32 tiles.

In all comparisons, we measure numerical differences with the Mean Squared Error (MSE) and visual differences by counting the number of Perceptually Different Pixels (PDP), as proposed by Yee [46]. This perceptual error metric acts in the Lab space and is plotted in black and blue.

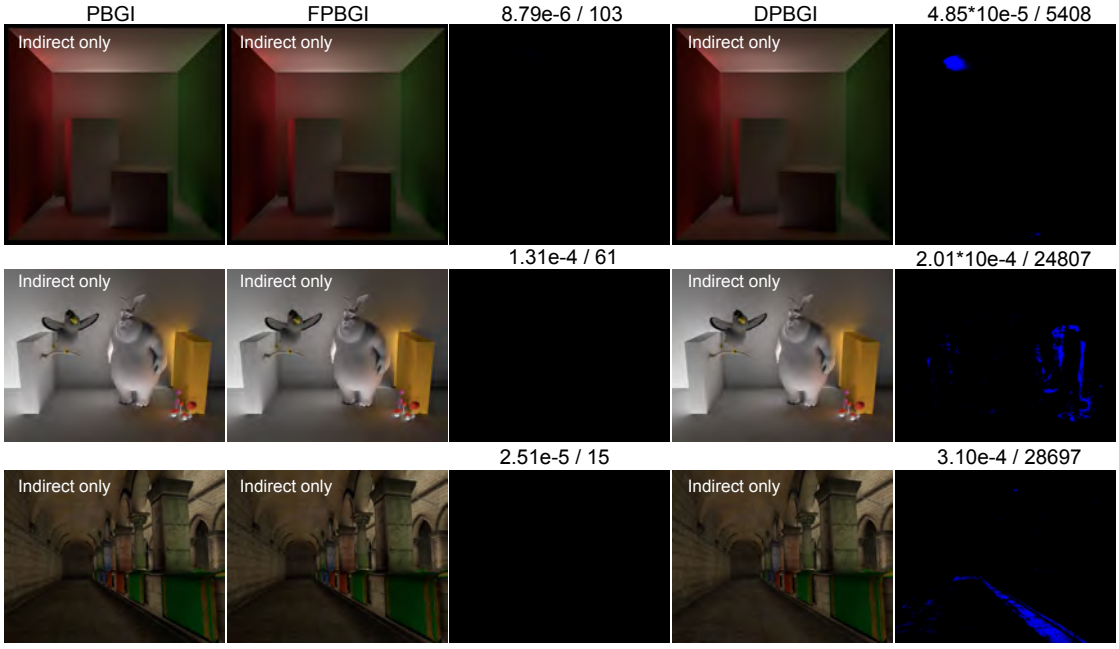


Figure 2 Error analysis on the indirect lighting contribution for FPBGI and DPBGI against PBGI. Perceptual differences [46] are plotted in black (no visible difference) and blue (visible difference). The MSE between RGB images and the number of Perceptually Different Pixels [46] (PDP) are indicated in the format <MSE>/<PDP> on top of difference images.

In Fig. 2, we compare FPBGI with the original PBGI algorithm on three different scenes. Overall, we observe a negligible error, both from a perceptual and numerical point of view. The original PBGI algorithm can indeed be trivially sped-up by reducing the resolution of the microbuffers (i.e., higher solid angle threshold in the tree traversal), which immediately translates into coarser cuts for each receiver and reduced rasterization time. Therefore, we also compare to such a *degraded* PBGI setting (DPBGI), with microbuffer resolution decreased so that the total rendering time is as close as possible to our FPBGI. In this case, DPBGI produces significantly stronger errors, with noticeable aliasing appearing.

In Table 3, we report timings and errors for the four different examples shown in Fig. 2 and Fig. 3. Here, we can assess the benefit of our factorized approach, with a speed-up ratio for the total rendering time (including BRDF evaluation and initial set up) ranging from 2.2 to 3.8 compared to the original PBGI algorithm. Looking at the specific portion of the algorithm that we target (rasterization), the speed-up ratio ranges from 3.4 to 5.2 for the

cut computation and from 1.4 to 3.2 for the micro-rasterization. In all cases the receiver clustering time is negligible.

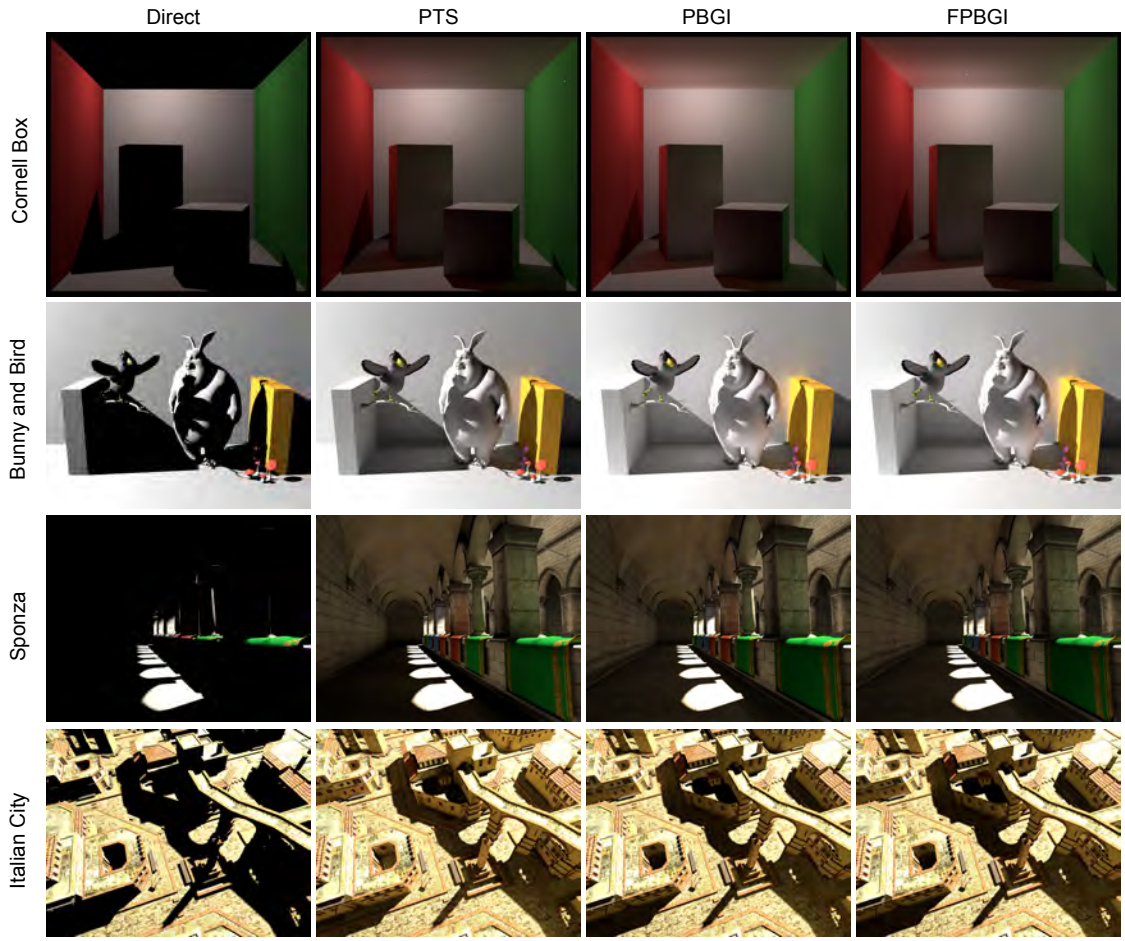


Figure 3 Visual comparison of final renderings.

In Fig. 3, we provide a visual comparison of the final rendering (direct+indirect illumination) between a fully path-traced solution (PTS), PBGI and FPBGI. We can observe that PBGI and FPBGI provide similarly good approximations of the PTS, which is typically an order of magnitude slower than FPBGI.

We also analyze the influence of the two main parameters of FPBGI: the number of clusters per-tile k and the far-near threshold ϵ. In Fig. 4, we illustrate their influence on the Sponza scene. We observe that the influence of k clearly dominates on the approximation quality, as measured by numerical and perceptual errors. However, looking closely at the result, we

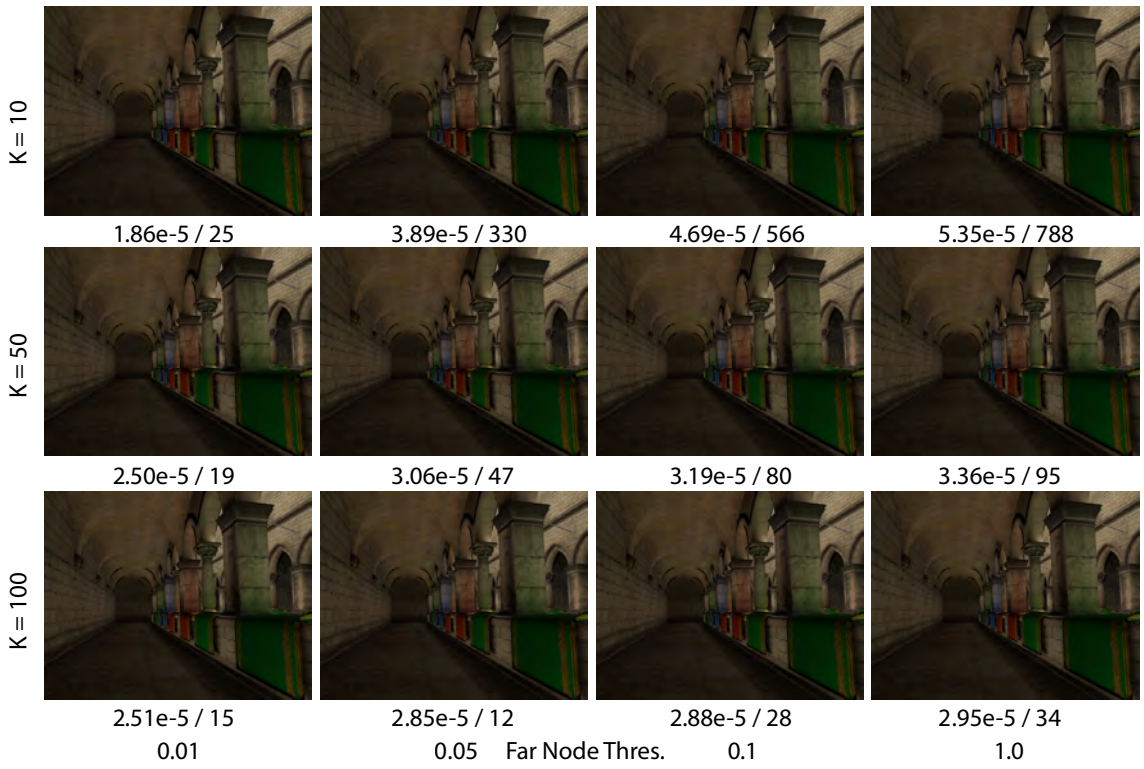


Figure 4 Parameter influence with <MSE>/<PDP> to the PBGI solution for each pair.

can see that, under a very small k value, large values of ϵ cause large visible artifacts. In Fig. 5, we plot the speed-up evolution under variations of these two parameters. We empirically determine $k = 100$ and $\epsilon = 10e^{-2}$ as good default values for all the scenes we experimented with. Last, with visually invisible differences, FPBGI inherits the temporal coherence of PBGI: we illustrate this behavior in an accompanying video with three sequences showing animated lighting, camera and models.

Discussion. Alternatively to our approximation technique, recent approaches [32, 36] propose to maximize the fine-grained parallelism of the PBGI algorithm in order to map it efficiently on GPU architectures. Clearly, our approach is orthogonal to such methods, but preserves the natural parallelism of PBGI. However, compared to such methods, an additional specific preliminary pass would be required to gather the shared microbuffers. As future work, at least two alternative solutions may be further developed to combine our factorization with an efficient GPU implementation: first, the typical number of clusters is

large enough to load the numerous GPU computing units with cluster cuts computations using a naive implementation (i.e., one thread per-cluster first, then one thread per-receiver); second, a more evolved solution could use the two-layer GPU computing model (blocks and threads) to make threads belonging to the same block define concurrently the cluster cut and microbuffers in shared memory before synchronizing them and letting them processing their receiver-specific components, using the ManyLoDs algorithm [36] at both stages.

Interestingly, Holländer et al. [36] proposed an acceleration exploiting *temporal* coherence only, the *lazy* scheme which reuses cuts over consecutive frames, while our factorized approach exploits *spatial* coherence. Combining both approaches could help exploiting *spatio-temporal* coherence to its full extent.

Our approach has two main limitations. First, the cluster cut may be over-conservative and the resulting per-receiver cut can be too refined. Although this does not influence the rendering quality, this remains sub-optimal. A solution could be to allow receivers to "walk-up" the tree while refining their cut. Second, the two main parameters of the algorithm have fixed values. These values could be optimized dynamically and vary spatially by using the PBGI tree to perform a quick scene analysis. Last, our approach can be seen as a simplified *hierarchy* of receivers. It would then be interesting to determine how to reformulate the PBGI algorithm to rasterize, adaptively, the PBGI tree against the receiver/pixel one to reach a fully adaptive solution.

5 Conclusion

We have proposed a factorized evolution of the PBGI algorithm which exploits spatial coherence to significantly speed up the computation of indirect diffuse illumination effects. By combining an initial variational clustering with per-cluster cuts and microbuffers, we showed that the individual receiver workload boils down to a local geometry rasterization followed by a microbuffer compositing. As a result, we obtain a speed-up ranging from 2x to 4x, without any visible image degradation. Our approach is easy to implement in any PBGI framework and has a reduced set of intuitive control parameters.

Acknowledgements. This work has been partially supported by the China Scholarship Council and 863 Program of China under Grant No. 2012AA01A306, the European Commission under contract FP7-287723 REVERIE and the ANR iSpace&Time project.

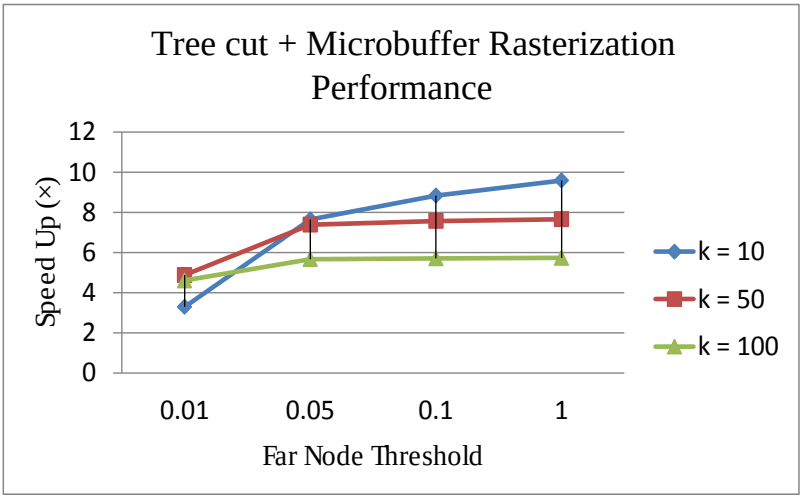


Figure 5 Parameters influence on the speed-up.

外文论文 Wavelet Point based Global Illumination

Abstract

Point-Based Global Illumination (PBGI) is a popular rendering algorithm in movie and motion picture productions. This algorithm provides a diffuse global illumination solution by caching radiance in a mesh-less hierarchical data structure during a pre-process, while solving for visibility over this cache, at rendering time, for each receiver, using a microbuffer, which is a localized depth and color buffer inspired from real time rendering environments. As a result, noise free ambient occlusion, indirect soft shadows and color bleeding effects are computed efficiently for high resolution image output and in a temporally coherent fashion. We propose an evolution of this method to address the problem of non-diffuse inter-reflections and refractions. While the original PBGI algorithm models radiance locally using spherical harmonics, we propose to use warped wavelets to better localize the radiance representation in the presence of highly directional reflectance, an importance-driven adaptive model for the per-receiver microbuffer to capture accurately the incoming radiance, further with a fast wavelet radiance product model to evaluate outgoing radiance from incoming radiance. We address the induced larger memory footprint by encoding hierarchically the wavelets in the PBGI hierarchy. In addition, a viewtree-based approach is introduced to compute the multiple bounces reflections / refractions. As a result, our rendering algorithm allows to handle non-lambertian BSDF in the light transport, reproducing caustics with a similar quality to bidirectional path tracing for only a fraction of the computation time. Our approach is simple to implement and easy to integrate in any existing PBGI framework, with an intuitive control on the approximation error. We evaluate it on a collections of example scenes.

1 Introduction

Over the last decade, *global illumination* (GI) has become a standard requirement for almost any industrial computer graphics production, from movie special effects to motions pictures and TV shows. Among the vast repository of rendering algorithms that supports (at

least a subset of) GI effects, Point-Based Global Illumination (PBGI) is certainly one of the most widely used solution. The basic idea of PBGI is to decorrelate the scenes complexity from the GI computation by substituting a shaded point clouds to the scene when computing any indirect lighting effect, keeping the polygonal representation for direct visibility from the camera only. This algorithm is particularly efficient at simulating a large number of the diffuse effects which are visually appealing in GI, including ambient occlusion, indirect soft shadows and color bleeding. The algorithm itself can be easily adapted to compute multiple light bounces, subsurface scattering and more general volumetric effects while staying fast, embarassingly parallel and finely tunable by artists. However, although final receiver surface samples may be lit using any BRDF, the indirect lighting effects captured by classical PBGI are restricted to diffuse inter-reflections and do not allow to reproduce caustics for instance, implying alternative GI solutions (e.g., Monte Carlo Ray Tracing, Photon Mapping) for these cases.

Principle The PBGI rendering algorithm [24] can be summarized in two main stages: *radiance caching* and *multiview rasterization*. During the first stage, the virtual scene is sampled with a dense point cloud, the point cloud is shaded from the primary light emitters and a hierarchical data structure (e.g., BVH) is generated to store it. The resulting *PBGI tree* can therefore be seen as an hierarchical "spatial cache" of the scene radiance. In the second stage, the indirect illumination of each receiver (e.g., unprojected image pixel) is evaluated by localizing a color+depth hemispherical buffer (*microbuffer* or MB) at the receiver position and filling it by rasterizing the PBGI tree onto it. This rasterization resembles the hierarchical Z-buffer algorithm, only substituting the PBGI tree to the actual scene filling a specific MB for each receiver. The mesh-less nature of the PBGI tree makes it easy to extract a receiver-dependent adaptive level-of-detail (LoD) of the scene as a "cut" in the tree, reducing drastically the computational footprint of the many rasterizations required to fill the MBs of all the pixels of a high resolution image. The final receiver radiance (e.g., pixel color) is obtained by convolving the filled MB with the receiver reflectance function.

Limitations One key feature of PBGI is to reproduce efficiently noise-free diffuse GI effects by modeling radiance in the PBGI tree nodes using spherical harmonics (SH). Unfortunately, even using a larger number of coefficients, SH are not able to capture high frequencies efficiently, which translates in our case to non-diffuse reflections or refractions. Consequently, caustics stemming from metals, plastics, glass and other reflective or refractive materials are not handled with classical PBGI frameworks. Even when ignoring the performance issue induced by a larger number of SH coefficients, ringing artifacts quickly appear and the second step of the algorithm remains flawed: as the receiver color is evaluated by convolving its BRDF with a discretized incoming radiance (i.e., the MB) the case of glossy to nearly specular reflections cannot cope with the typical low resolution of the MB and again, the intrinsic speed of PBGI vanishes when increasing the MB size, causing additional artifacts as well.

Overview In this paper, we introduce *Wavelet PBGI* (or WPBGI) to address efficiently the problem of non-diffuse GI effects such as caustics. Basically, we propose a wavelet radiance model to capture out-going radiance at each node of the PBGI tree using Haar wavelets warped over the space of directions (sphere). Such a basis function can capture efficiently the localized regions with high frequency out-going radiance. We introduce an adaptive model for the MBs (extend to spherical buffer to support refractions) to evaluate incoming radiance, allowing to guide the LoD extraction based on the classical geometric factor (distance, incidence angle) but also on appearance parameters, such as the estimated incoming radiance or the BSDF glossiness for instance. A fast wavelet radiance product model is further proposed to support fast outgoing radiance computation from incoming radiance and BSDF. The wavelet model induces a significantly larger memory footprint, which is solved by storing the nodes radiance wavelets hierarchically, expressing nodes items w.r.t. to their (average) parent one. Such an hierarchical representation also allows to compute adaptive importance-driven cuts when rasterizing for a particular MB at rendering time. Therefore, as these cuts exhibit very fine structures (e.g., specular spots) next to coarser ones, we introduce an adaptive model for the MBs, allowing to guide the LoD

extraction based on the classical geometric factor (distance, incidence angle) but also on appearance parameters, such as the estimated incoming radiance or the BRDF glossiness for instance. We also propose a view-tree based approach to accelerate the multiple bounces computation by splatting the point cloud tree to itself rather than traversing and splatting for each point of the point cloud. Our improved PBGI algorithm allows to render images with caustics effects and all flavors of indirect non-diffuse effects, up to close to perfectly specular materials, and also refracted materials. To summarize, we make the following technical contributions:

1. wavelet out-going radiance representation for each node of the PBGI tree,
2. importance-driven adaptive microbuffers for high frequency incoming radiance and reflectance.
3. hierarchical coding (wavelet-like) of the PBGI tree nodes,
4. view tree based multiple bounces computation model.

2 Previous Work

PBGI. PBGI was first proposed by Christensen [24] to evaluate diffuse lighting transport with mesh-less hierarchy and microbuffers. The notion of point-based substitutes was introduced by Bunnell [77] for real time ambient occlusion and indirect illumination. Ritschel et al. [32] then substituted the warped microbuffer for the cube ones to allow for importance sampling based on the BRDF of receivers, with an efficient GPU implementation. Holländer et al. [36] proceeded with fine-grained parallelism of the adaptive cut computation. A number of approaches have been proposed for the cut computation, such as Maletz and Wang [35] who used an importance-driven point projection based on an initial clustering, Tabellion [37] who presented a cut picking algorithm for HDR imaging, and Wang et al. [34] who proposed a factorized approach by reusing the tree-cut and microbuffers based on spatial coherence. The PBGI memory issue has been tackled by Kontkanen et al. [31], who proposed an out-of-core framework for PBGI with cache-coherent tree construction and traversal, and by Buchholz and Boubekeur [33] who proposed an in-core solution by quantizing all tree nodes against the resulting look-up table.

Wavelets based Rendering. Wavelets have been used to represent light transport functions [49, 50] and [55], to approximate environment map [52] and to represent multiple functions involving lighting, BRDF, local visibility [53, 54]. We use 2D Haar basis for simplicity, but it would be interesting to use a more sophisticated scheme, such as spherical wavelets by Schröder and Sweldens [66].

Other Basis Functions. (Hemi)Spherical harmonics have been used to store irradiance environment map [59], transfer function and the lighting function [51], incoming radiance [42] and outgoing radiance [24]. Spherical harmonics can reconstruct low-frequency functions efficiently, but for high frequency BRDF or lighting, they require a large number coefficients, and also suffer from the "ringing" problem. Spherical Gaussians have also been widely used to represent light transport functions [64, 60, 78] and BRDFs such as [62, 63]. Lehtinen et al. [79] proposed a hierarchical mesh-less basis to represent diffuse light transport.

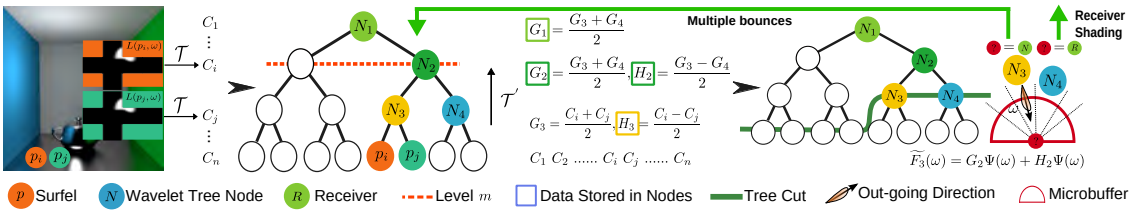


Figure 6 Principle. Starting from point cloud sampling, the points or surfels are shaded with directional distribution, by sampling the radiance distribute function according to a cube map then haar wavelet transforming ($\mathcal{T}$) to coefficients, such as p_i and p_j with C_i and C_j to represent the out-going radiance. The point cloud is further organized in to a spatial hierarchy: wavelet tree(the middle), with the radiance distribution of the internal node evaluated by wavelet analysis ($\mathcal{T}'$) on the surfels' coefficients vector. The point cloud tree is traversed and splatted to itself to compute the multiple bounces or for a receiver to shade it.

Table 2 Notation used in the paper.

x, n	Point position and normal
L_o, L_i	Outgoing radiance and incoming radiance
ρ	BRDF
ω_i, ω_o	Local incident, outgoing directions
V	Binary visibility
$\mathcal{T}_2, \mathcal{T}_4$	2D and 4D wavelet transform
U	Coeffs. with 4D wavelet transform of $\rho(\omega_i, \omega_o)$
D	Coeffs. with 2D wavelet transform of $\rho(\omega_o)$
W	Coeffs. with 2D wavelet transform of $L_i(\omega_i)$
L_{oj}, F_j	Original outgoing radiance of point / node j
$\widetilde{L_{oj}}, \widetilde{F_j}$	Reconstructed outgoing radiance of point / node j
Ψ	Haar wavelet basis function
C_j	Coeffs of point j in wavelet expansion of L_{oj}
G_j, H_j	Average and detail Coeffs vector of node j
j^-, j^+	Index of the left and the right child of node j
P_j	Path from the root to the parent of node j
s_j	Sign of node j: 1 left, -1 right
m	Prescribed level
$\mathcal{F}, \mathcal{F}_\rho, \mathcal{F}_l$	Total, BRDF and lighting importance function
γ	Radiance threshold
M	Number of pixels of the microbuffer
ω_k	Direction of the kth pixel in the microbuffer

3 Wavelet PBGI

3.1 Super Point Model

Each point in the point cloud or the node in point cloud tree is a sender, as the direct outgoing radiance for them is stored to compute the indirect illumination of receivers. In the case of multiple bounces or environment mapping, these points or nodes are also receivers. We propose a super point model to describe such a point or node, which includes modeling the outgoing radiance with Haar wavelets, evaluating the incoming radiance and visibility with spherical importance driven microbuffer, further with a wavelet radiance product model to compute outgoing radiance from incoming radiance.

The radiance equation with direct lighting is as

$$L_o(x, \omega_o) = \int_{\Omega_{2\pi}} L_i(x, \omega_i)\rho(x, \omega_i, \omega_o)V(x, \omega_i)(\omega_i \cdot n)d\omega_i. \quad (1)$$

Wavelet Radiance Model To properly capture the localized reflectance of non-diffuse surfaces, we propose to model it in the point samples using Haar wavelets instead of SH. The radiance distribution is parameterized as $L_o(x, \omega_o)$. At caching time, the radiance distribution is evaluated for a fixed number of points, reducing the global radiance model to a collection of two-dimensional functions $L_{oj}(\omega)$, where j is the index of the surfel. Each of these function is projected onto the Haar basis $\Psi = \{\psi_i\}$ by writing it as a series of expansion,

$$L_{oj}(\omega) = \sum_i c_i \psi_i(\omega),$$

with $C(j) = \{c_i\}$ the vector of coefficients in the basis. To evaluate the coefficients, $L_{oj}(\omega)$ is sampled in a cube map yielding six small (e.g., 32×32) images for each surfel. We then Haar-transform each image and store the coefficients for each surfel with a tree [54], which allows on-demand pointwise reconstruction (i.e., compressed radiance estimation at the surfel for a given direction) avoiding to decompress the entire wavelet. This compressed radiance $\widetilde{L}_{oj}(\omega)$ is reconstructed as follows:

$$\widetilde{L}_{oj}(\omega) = C_j \Psi(\omega).$$

Wavelet Radiance Product The BRDF of a point can be parameterized as $\rho(\omega_i, \omega_o)$ in local coordinate (the z axis is the same as the normal direction) and then be represented by 4D wavelet coefficients as follows.

$$U = \mathcal{T}_4 \rho(\omega_i, \omega_o). \tag{2}$$

The incoming radiance of a point can be wavelet transformed either, resulting in 2D coefficients,

$$W = \mathcal{T}_2 L_i(\omega_i). \tag{3}$$

We observe the following formulas exist:

$$D = UW, \quad L_o = \mathcal{T}_2^{-1} D \tag{4}$$

where D represents the result by doing production between the BRDF wavelet coefficients U and the incoming radiance wavelet coefficients W, actually D is the outgoing radiance

wavelet coefficients with 2D wavelet transform, so the outgoing radiance L_o can be evaluated by doing an inverse transform $\mathcal{T}_2^{-1}$ for D. As D is in local coordinate, it is decode at first from the frequency domain to space domain, sampled in a cube map and then wavelet transformed.

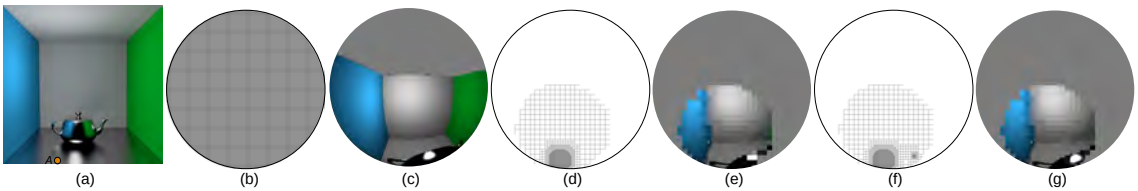


Figure 7 An illustration of the importance driven MB construction. Start from a glossy receiver denoted as A in (a), the uniform MB is shown in (b) and (c) and the adaptive MB is shown in (d),(e),(f) and (g). (b) visualizes the resolution of each pixel of the MB, while the splatted one is visualized by (c). (d) shows the resolution of the initial BRDF AMB, and after the nodes splatting shown in (e). After the lighting adaptive, the level visualization is shown in (f), and (g) denotes the nodes splatting results after refining according to the lighting driven factor.

3.2 Importance Driven Microbuffer In the case of the PBGI with MB, (1) turns into

$$L_o(x,\omega_o) = \sum_{k=0}^{M} F_j(\omega_k)\rho(x,\omega_k,\omega_o)V(x,\omega_k)(\omega_k \cdot n). \tag{5}$$

As the $V(x,\omega_k)$ is solved by the MB, we ignore this term, and obtain a simple form from (5)

$$\widetilde{L_o} = \sum_{k=0}^{M} KF_j\rho, \tag{6}$$

where K represents the other terms.

According to (6), we propose to construct MBs driven by an importance function considering both the BRDF and the lighting, resulting in adaptive MBs (AMBs). When the point is the view receiver (the outgoing direction is fixed), the importance function of pixel k is represented as $\mathcal{F}(k) = \mathcal{F}_l(k)\mathcal{F}_\rho(k)$.

The computation of $\mathcal{F}\rho$ is similar to the method used in [32] but with subdivision instead of warping, and $\mathcal{F}_l$ is defined as

$$\mathcal{F}_l(k) = \begin{cases} 1, & \frac{L_i(k)}{d^{\Delta_k}} > \gamma; \\ 0, & otherwise, \end{cases}$$

where d is the dimension of the division, Δ_k is the incremental of the pixel k since starting traversal.

When the point is not a view receiver, the importance function becomes $\mathcal{F}(k) = \mathcal{F}_l(k)$.

In our implementation, AMBs are spherical buffers (to simulate refraction) and organized as trees. Each buffer node in an AMB tree has fixed number of pixels (eg. 2×2).For each receiver, a MB is initialized uniformly (e.g. 16×16), refined by $\mathcal{F}_\rho$ and further updated by $\mathcal{F}_l$ during traversing and splatting. Note that the nodes splatted to the MB pixels are all recorded respectively instead of only the nodes with nearest distance. If a pixel is subdivided, all the nodes recorded in this pixel will be re-traversed and re-splatted. After the traversing and splatting step, the near leaf nodes are processed in a post raycast step resulting in the final MB filled with incoming radiance.

3.3 Hierarchical Wavelet Coding

With our radiance compression scheme in hand, we can build the PBGI tree in a bottom-up fashion as a complete binary bounding sphere hierarchy, with the its leaves formed by the sampled surfel set and the internal nodes averaging their related subtree attributes, including position, normal, size and radiance distribution. Since our wavelet radiance representation at each surfel (i.e., tree leaf) is expressed in the global coordinate system and the Haar wavelets support linear operators, the radiance approximation for an internal node j can be expressed directly by averaging its children coefficients:

$$C_j = \frac{C_{j^-} + C_{j^+}}{2}.$$

Taking inspiration from progressive coding schemes, our key idea to reduce the memory footprint of our wavelet PBGI tree is to encode the tree **itself** in a wavelet fashion, by expressing the radiance wavelet coefficient vector of a node w.r.t. to the coefficient vector

of the parent node. In the case of binary tree such as our PBGI tree, this boils down to a 1D Haar transform decomposing C_j into an average vector $G_j = (G_{j^-} + G_{j^+})/2$ and a detail vector $H_j = (G_{j^-} - G_{j^+})/2$. We At caching time, we can retain only G_0 (the root average coefficient vector), discard the other average vectors and store in the nodes only the detail vectors, ignoring the ones with a L^2 norm smaller than a user-defined compression threshold. To avoid storing the entire list of nodes vectors at any intermediate state, we compute this compressed representation during a **post-order depth-first** traversal of the PBGI tree.

At rendering time, we can reconstruct the radiance coefficient vector of a given node j as:

$$C_j := G_{\text{root}} + \sum_{k \in P_j} s_k H_k.$$

Since wavelet compression provides a spatialize representation, evaluating radiance in a given direction does not require recompressing the full coefficient vectors.

Assuming a full wavelet decomposition of the tree has obvious impact on reconstruction time. This pitfall can be strongly diminished by performing the compression only partially, retaining the average vector up to prescribed level m. This allows reconstructing the radiance in constant time for any node upper than level m and significantly shrinking the reconstruction time for the other nodes by boot-strapping the reconstruction directly from the level-m ancestor r of the node:

$$C_j := G_m + \sum_{k \in \{P_j \setminus P_r\}} s_k H_k$$

3.4 Multiple Bounces with the View-Tree

The points in the point cloud are treated as receivers during multiple bounces. Based on the observation that some nodes from the point cloud tree contribute similarly to the points included in other nodes, we propose a view-tree based solution to accelerate the traversal and splatting process that means the point cloud tree is traversed and splatted for itself rather than for each point in the point cloud.

Let's start with a pair N_i/N_j, where N_i is called a *sender node* and N_j is a *receiver node*. The purpose of our approach is to find these pairs which satisfy N_i contributes **similarly** to

Table 3 Performance measures.

		BPT		WPBGI						Error
scene	max path.	num. samples	total (h)	pts. (M)	ω res	memory (GB)	pr. time (m)	render time (h)	total time (h)	*MSE*
Ring	3	16384	5.49	5	128^2	5.98	3.22	1.14	1.20	4.589e-5
Corner	3	16384	6.63	8	32^2	6.62	3.05	1.57	1.63	1.570e-4
Bunny	3	16384	10.53	4	32^2	5.68	12.28	0.84	1.04	6.527e-5
Kitchen	3	16384	11.20	5	32^2	5.26	6.57	3.66	3.77	7.948e-5
Sibenik	3	16384	10.33	10	32^2	5.85	11.07	1.75	1.94	2.099e-4
Sphere	5	16384	22.80	2	32^2	5.06	28.80	0.95	1.43	5.370e-4
Torus	5	65536	40.01	1	32^2	6.53	201.61	1.01	4.37	3.116e-4

all the points contained in N_j.. The solid angle between the sender nodes and the receivers used to determine the tree cut is called *sender solid angle*. The solid angle between the view node and the center of the sender node is used to determine whether the sender node contributes **similarly**, called *view solid angle*.

The point cloud tree is traversed for itself, starting from N_0/N_0, where N_0 is the root of the point cloud tree. If the view solid angle is larger than a preset threshold, the sender node will be traversed for all the children nodes of the current view node. Otherwise, the sender node will traversed for the view node until the sender solid angle condition satisfies, and all the nodes and related data (direction and distance) are stored in the view node and also splatted to the view node's MB. When no sender node required to be traversed for the current view node, the children of the view node will be processed. When the leaf view node is reached, all the nodes records during the path to the view tree root which generate the tree cut are processed hierarchically. If the leaf node's normal is similar to the normal its ancestor node's, then it can reuse the ancestor node's MB, otherwise, the nodes in the ancestor node are required to be splatted the MB of current leaf view node.

The MB of a leaf node filled with incoming radiance is wavelet 2D transformed to obtain the wavelet coefficients, which are multiplied with BRDF coefficients to get the outgoing radiance with formula 4. After the outgoing radiance of the leaf sender node is updated, the corresponding radiance coefficients of the whole point cloud tree are updated in the post-order to keep low memory usage.

4 Results

We implemented our technique in the Mitsuba Renderer [2]. Comparisons are performed against the original PBGI algorithm [24], the progressvie photon mapping [18] and PBGI with SH; performances are measured on a 2.67GHz Intel i7 (8 cores) with 9GB of main memory. Images are rendered at a 1024×768 pixels resolution (except for the Ring scene, the Cornel Box scene and the Sphere scene, at 1024×1024) with 32×32 tiles, with adaptive sampling for anti-aliasing. In all comparisons, we measure numerical differences with the Mean Squared Error (MSE). The BSDFs in all the example scenes are "roughconductor" for glossy reflected materials and "roughdielectric" for refracted materials, which both implement realistic microfacet scattering models.

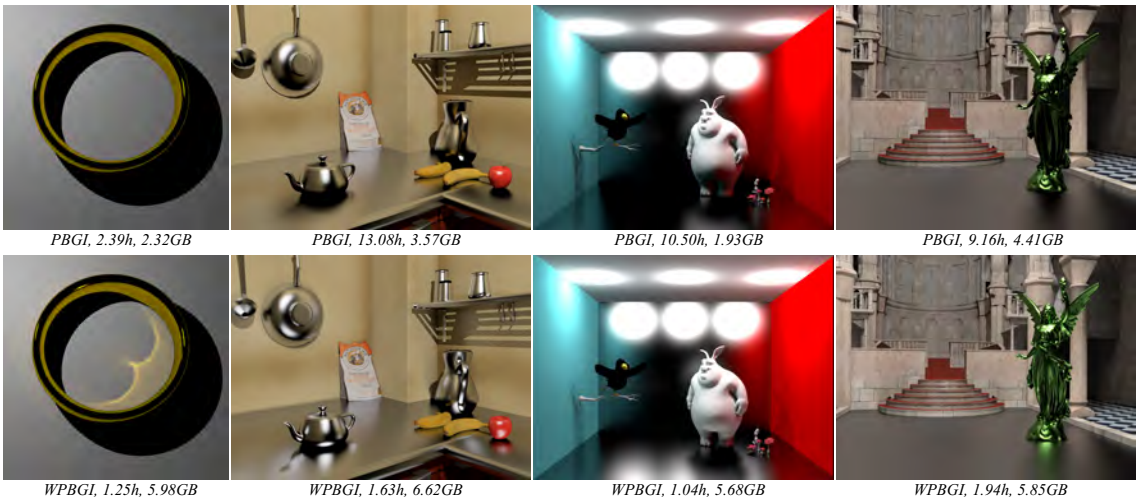


Figure 8 Comparison between PBGI and WPBGI.

Comparison between PBGI and WPBGI. We compare our WPBGI with PBGI in Fig. 8. We can observe WPBGI can simulate caustics effect (non-diffuse light transport), while PBGI only supports diffuse lighting transport. The WPBGI approach can also obtain times of speed-up ratio for the total rendering time compared with PBGI because of the usage of importance driven microbuffer.

Figure 9 Comparison with spherical harmonics based PBGI. (a) BPT, 2048 samples per pixel as the ground truth (b) WPBGI, with cube map resolution $6 \times 32 \times 32$, and discard threshold 0.002 (c) PBGI based on spherical harmonics with 64×3 coefficients and (d)PBGI based on spherical harmonics with 256×3 coefficients for radiance caching of the glossy surfels and the nodes and (e) PBGI. The numbers below the images represent the total rendering time and the peak memory usage.

Comparison between SH based PBGI and WPBGI. By comparing our WPBGI with the spherical harmonics based solution in Fig 9. we observe spherical harmonics based PBGI can not represent sharp features without large number of coefficients (even 256×3 coefficients are insufficient), while our wavelet based solution has improvement on both performance and memory.

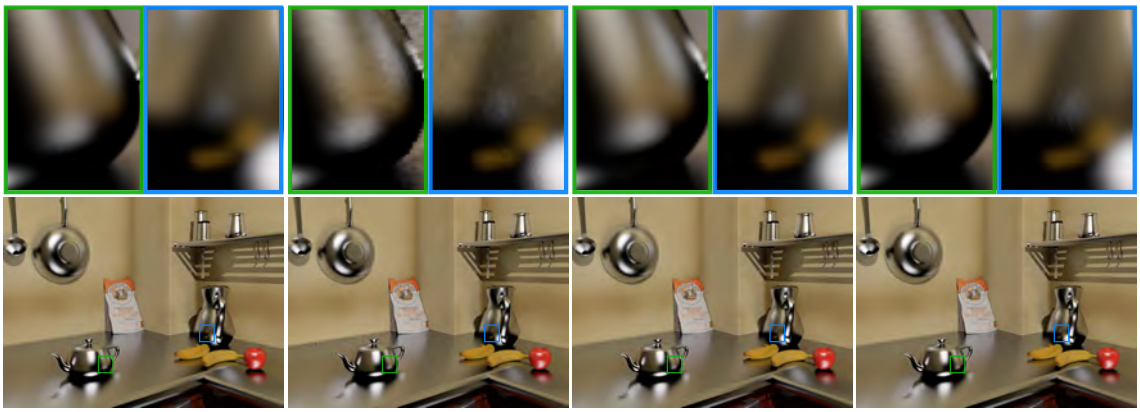
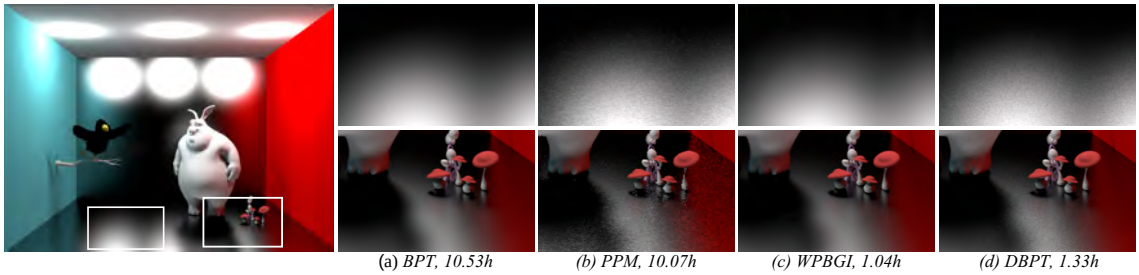


Figure 10 Corner scene comparison: Visual comparison of final renderings.

Comparison between WPBGI and Other Techniques. We verify the accuracy and visual quality of the images rendered with our wavelet based algorithm. In Fig. 11, Fig. 12, and Fig. 13, we provide a comparison between a bidirectional path-traced solution (BPT),

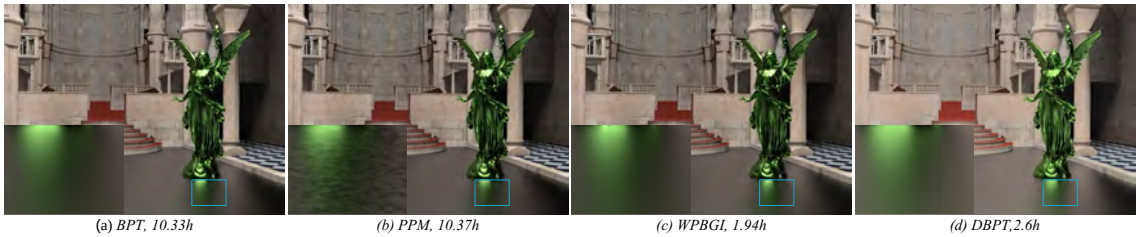


(a) *BPT, 10.53h* (b) *PPM, 10.07h* (c) *WPBGI, 1.04h* (d) *DBPT, 1.33h*

Figure 11 Bunny scene comparison: Visual comparison of final renderings.

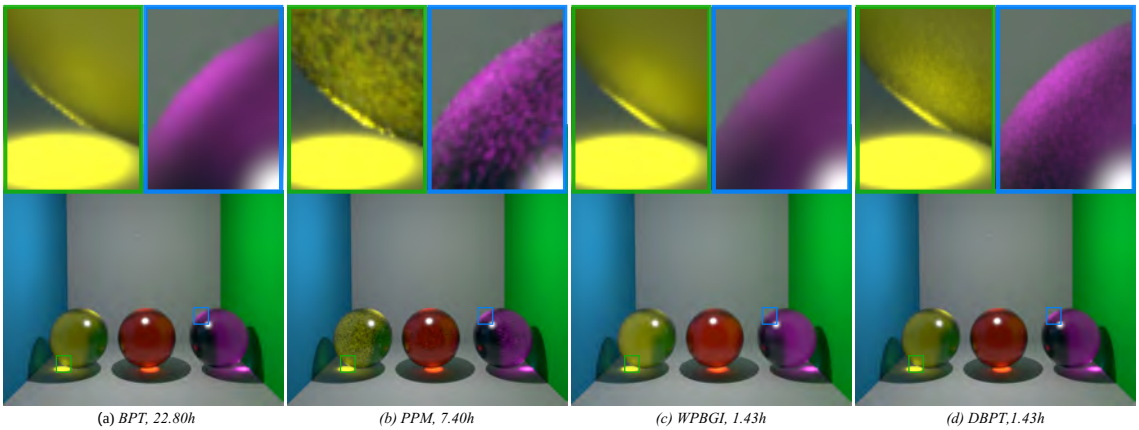
(a) *BPT, 11.2h* (b) *PPM, 11.55h* (c) *AWPBGI, 3.77h* (d) *DBPT, 5.56h*

Figure 12 Kitchen scene comparison: Visual comparison of final renderings.



(a) *BPT, 10.33h* (b) *PPM, 10.37h* (c) *WPBGI, 1.94h* (d) *DBPT,2.6h*

Figure 13 Sibenik scene comparison: Visual comparison of final renderings.



(a) *BPT, 22.80h* (b) *PPM, 7.40h* (c) *WPBGI, 1.43h* (d) *DBPT,1.43h*

Figure 14 Sphere scene comparison.

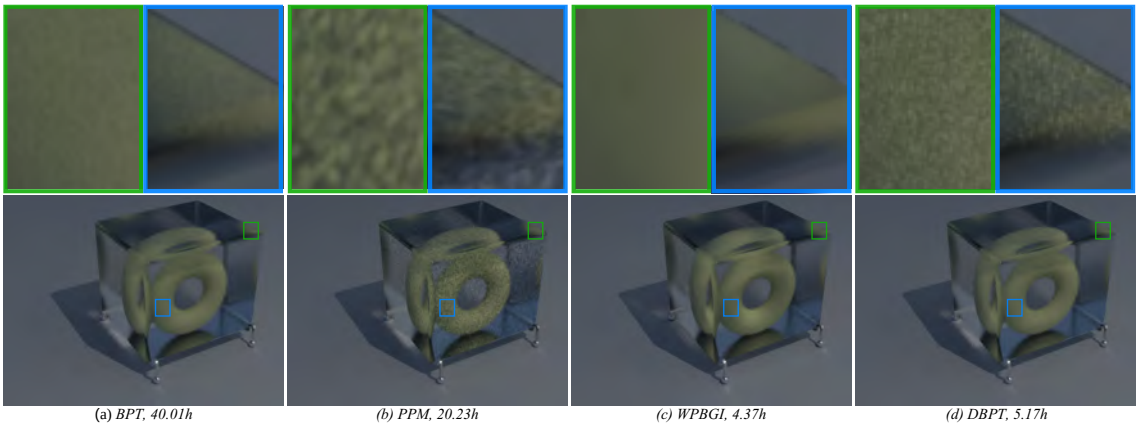


Figure 15 Torus scene comparison.

progressive photon mapping (PPM), and WPBGI. We can observe that WPBGI provides similarly good approximations of BPT, which is typically an order of magnitude slower than WPBGI, while the results of the PPM suffer form noise with the same rendering time as the BPT. We also compare to a *degraded* BPT (DBPT) solution with less samples but the similar rendering time as WPBGI, which generates noisy results.

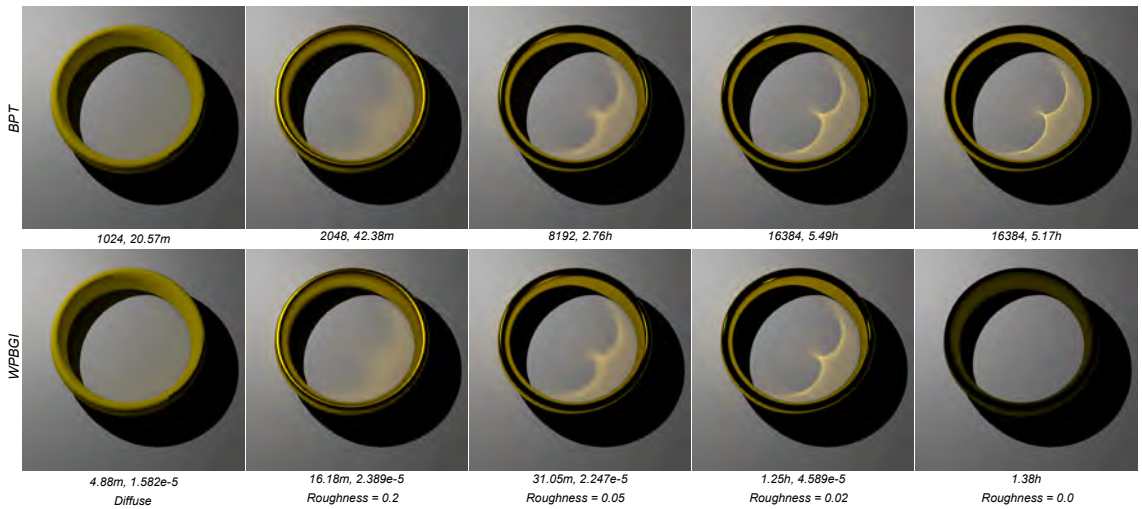


Figure 16 Ring scene comparison for all frequency BRDF between back ground (BPT) and WPBGI. The numbers below the top images are the number of the samples and the total rendering time. Then numbers below the bottom images are the total rendering time and the *MSE* error compared with the references, while the value of the *roughness* means the glossiness of the BRDF (*roughconductor*).

In Fig. 16, we verify the accuracy of the images produced by our WPBGI when varying the glossiness of the BRDF. Artifacts free images are produced with times of speed up compared with the references (created by BPT solution) except the last one in which the BRDF is specular reflection.

Performance and Timings. In Table 3, we report timings and errors for the five different examples shown in Fig. 10, Fig. 11, Fig. 12, Fig. 13 and Fig. 16 (glossiness 0.02). The resolutions of the cube map (ω *res*) to sample the out-going radiance are given as the size of a single face of the cube map and the *memory* is the peak memory usage during the whole process. The *pre. time* means the time cost by point cloud generating and wavelet tree constructing, while the *render time* includes the microbuffer initialization, wavelet tree traversal, nodes splatting and the BRDF convolving time. Here, we can assess the benefits of our approach, with a speed-up ratio for the total time ranging from 2.9 to 10.1 compared to the BPT algorithm, while the error is negligible.

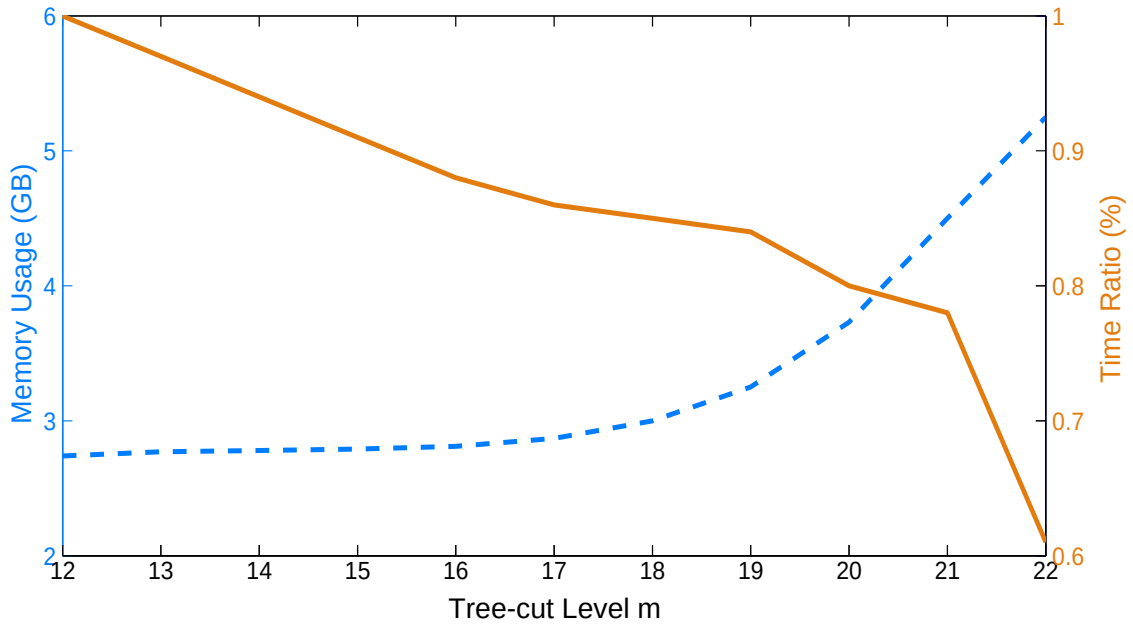


Figure 17 Memory usage (blue) and rendering time ratio (normalized to the level-12 timings) (orange) according to different tree cut level *m*.

We analysis the main parameters used in our algorithm. Fig. 17 summarizes the statistics of the memory usage of the wavelet tree and the times of scene corner with 5M pts by varying the value of m. From the data in the graph, we can conclude our squared wavelet approach can decrease the memory usage effectively as we decrease the value of m until some level (level 16). The reason why the curve becomes smooth after this level is that the detail coefficients becomes larger resulting less discards. As we increase m, our algorithm becomes faster but with more memory usage. By balancing these two factors, we set m to 20.

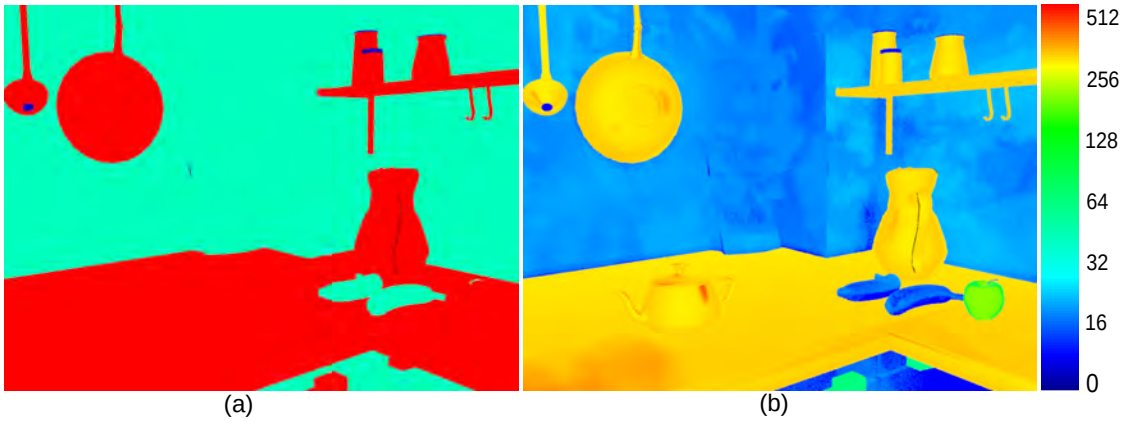


Figure 18 Comparison of the average resolution of microbuffers for corner scene between WPBGI with no AMB and WPBGI with AMB.

We also compare the performance for WPBGI between a non-adaptive scheme and adaptive scheme by visualizing the average resolution of the microbuffers in Fig. 18. The adaptive scheme costs 1.63h hours while the non-adaptive scheme costs more than 24 hours.
WPBGI inherits the temporal coherence of PBGI: we demonstrate this behavior in an accompanying video showing animated models.

Discussion. One limitation of our method is that the resolution of the cube map to sample the out-going radiance is set by users, not in an adaptive scheme. Another limitation is the metric to drive the lighting adaptive microbuffer. Now we use the value of the radiance to decide whether to subdivide a pixel of the microbuffer, however, sometimes high radiance value does not means a sharp lighting means the high resolution in this case is not required.

We have also try other metrics such as the difference of the radiance and the level of the node coefficients in the wavelet, but these metrics do not work well.

One cell is determined to be important cell when some nodes recorded in this cell have large radiance. However, this node may not contribute to the final result when it is totally occluded by others. In this case, using a high resolution for this cell has no improvement for the quality while increases the time cost.

Finally, we use haar wavelet to represent out-going radiance by sampling a global cube map, but the microbuffer we use is a local sphere buffer, which makes it not elegant when computing the outgoing radiance coefficients, so we are very interested to try the spherical wavelets by Schröder and Sweldens [66].

5 Conclusion

We have proposed an wavelet based solution of PBGI algorithm to simulate non-lambertian BSDF in the light transport including a wavelet radiance model to represent the outgoing radiance of the point cloud tree node, an importance driven microbuffer model to capture incoming radiance and a wavelet product model to evaluate outgoing radiance from incoming radiance and BSDF. We also present a view tree based approach to compute the multiple bounces. As a result, we can capture the caustics effects with similar quality as BPT, while obtain a speed-up ranging from 3x to 10x, without any visible image degradation. Our approach is easy to implement in any PBGI framework and has a reduced set of intuitive control parameters.